



Perl

Practical Extraction and Report Language



Larry Wall

Born	September 27, 1954 (age 60) Los Angeles, California , US
Nationality	American
Alma mater	Seattle Pacific University UC Berkeley
Occupation	Computer programmer, author
Known for	Perl
Spouse(s)	Gloria Wall
Children	4
Website	www.wall.org/~larry/

Perl is a programming language developed by Larry Wall, especially designed for text processing.

It stands for **Practical Extraction and Report Language**.

It runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

Perl

"Practical Extraction and Reporting Language" written by Larry Wall and first released in 1987

Perl has become a very large system of modules

name came first, then the acronym

designed to be a "glue" language to fill the gap between compiled programs (output of "gcc", etc.) and scripting languages

"Perl is a language for easily manipulating text, files and processes": originally aimed at systems administrators and developers

What is Perl

Perl is a High-level Scripting language

Faster than sh or csh, slower than C

No need for sed, awk, head, wc, tr, ...

Compiles at run-time

Available for Unix, PC, Mac

Best Regular Expressions on Earth

What's Perl Good For?

Quick scripts, complex scripts

Parsing & restructuring data files

CGI-BIN scripts

High-level programming

Networking libraries

Graphics libraries

Database interface libraries

What's Perl Bad For?

Compute-intensive applications (use C)

Hardware interfacing (device drivers...)

Perl Features

Database integration interface DBI supports third-party databases including Oracle, Sybase, Postgres, MySQL and others.

Perl works with HTML, XML, and other mark-up languages.

Perl supports both procedural and object-oriented programming.

Perl interfaces with external C/C++ libraries through XS or SWIG.

Perl is extensible. There are over 20,000 third party modules available from the Comprehensive Perl Archive Network (CPAN).

The Perl interpreter can be embedded into other systems.

Executing Perl scripts

- "bang path" convention for scripts:
 - can invoke Perl at the command line, or
 - add `#!/public/bin/perl` at the beginning of the script
 - exact value of path depends upon your platform (use "which perl" to find the path)

- one execution method:

```
% perl  
print "Hello, World!\n";  
CTRL-D  
Hello, World!
```

- preferred method: set bang-path and ensure executable flag is set on the script file

Perl Basics

- Comment lines begin with: `#`
- File Naming Scheme
 - *filename.pl* (programs)
 - *filename.pm* (modules)
- Example prog: `print "Hello, World!\n";`

Perl Basics

Statements must end with semicolon

```
$a = 0;
```

Should call `exit()` function when finished

Exit value of zero means success

```
exit (0);    # successful
```

Exit value non-zero means failure

```
exit (2);    # failure
```

Perl Basics

- Open a text editor and type the below and save as first.pl

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

```
# This will print "Hello, World"
```

```
print "Hello, world\n";
```

Perl Basics

- Open a text editor and type the below

```
C:\> perl first.pl
```

This execution will produce the following result –

```
Hello, world
```



Hooray!

CONGRATS!

Woo-Hoo!

Perl Basics

- Open a text editor and type the below

```
C:\> perl first.pl
```

This execution will produce the following result –

```
Hello, world
```

Perl Basics

- Comments in perl

```
# This is a single line comment  
print "Hello, world\n";
```

```
=begin comment
```

This is all part of multiline comment.

You can use as many lines as you like

These comments will be ignored by the compiler until the next =cut is encountered.

```
=cut
```


Perl Basics

- Single and Double Quotes in Perl

You can use double quotes or single quotes around literal strings as follows –

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

```
print "Hello, world\n";
```

```
print 'Hello, world\n';
```

This will produce the following result –

```
Hello, world
```

```
Hello, world\n$
```

Perl Basics

- Another example

```
$a = 10;
```

```
print "Value of a = $a\n";
```

```
print 'Value of a = $a\n';
```

This will produce the following result –

```
Value of a = 10
```

```
Value of a = $a\n$
```

Perl Basics

- "Here" Documents – double quoted

```
$a = 10;
```

```
$var = <<"EOF";
```

This is the syntax for here document and it will continue until it encounters a EOF in the first line.

This is case of double quote so variable value will be interpolated. For example value of a = \$a

EOF

```
print "$var\n";
```

Perl Basics

- "Here" Documents – single quoted

```
$a = 10;
```

```
$var = <<'EOF';
```

This is case of single quote so variable value will not be interpolated. For example value of a = \$a

```
EOF
```

```
print "$var\n";
```

Perl Basics

- Escaping Characters

Perl uses the backslash (\) character to escape any type of character that might interfere with our code. Let's take one example where we want to print double quote and \$ sign –

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

```
$result = "This is \"number\"";
```

```
print "$result\n";
```

```
print "\$result\n";
```

Perl Datatypes

- Escaping Characters

Perl uses the backslash (\) character to escape any type of character that might interfere with our code. Let's take one example where we want to print double quote and \$ sign –

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

```
$result = "This is \"number\"";
```

```
print "$result\n";
```

```
print "\$result\n";
```

Perl Datatypes

Integer

25 750000 1_000_000_000
8#100 16#FFFF0000

Floating Point

1.25 50.0 6.02e23 -1.6E-8

String

'hi there' "hi there, \$name" qq(tin can)
print "Text Utility, version \$ver\n";

Perl Datatypes

Boolean

0 0.0 "" "0"	represent False
all other values	represent True

Perl Datatypes

Types and Description

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, which we will see in the upcoming chapters.

Arrays –Arrays are ordered lists of scalars that you access with a numeric index which starts with 0. They are preceded by an "at" sign (@).

Hashes –Hashes are unordered sets of key/value pairs that you access using the keys as subscripts. They are preceded by a percent sign (%).

Perl Datatypes

Numeric Literals

Type	Value
Integer	1234
Negative integer	-100
Floating point	2000
Scientific notation	16.12E14
Hexadecimal	0xffff
Octal	0577

Perl Datatypes

String Literals

scape sequence	Meaning
\\	Backslash
\'	Single quote
\"	Double quote
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\Onn	Creates Octal formatted numbers

scape sequence	Meaning
\xnn	Creates Hexideciamal formatted numbers
\cX	Controls characters, x may be any character
\u	Forces next character to uppercase
\l	Forces next character to lowercase
\U	Forces all following characters to uppercase
\L	Forces all following characters to lowercase
\Q	Backslash all following non-alphanumeric characters
\E	End \U, \L, or \Q

Perl Datatypes

This is case of interpolation.

```
$str = "Welcome to \nericsonbatch!";  
print "$str\n";
```

This is case of non-interpolation.

```
$str = 'Welcome to \nericsonbatch!';  
print "$str\n";
```

Only W will become upper case.

```
$str = "\uwelcome to ericsonbatch!";  
print "$str\n";
```

Perl Datatypes

Whole line will become capital.

```
$str = "\UWelcome to ericsonbatch!";  
print "$str\n";
```

A portion of line will become capital.

```
$str = "Welcome to \Uericsonbatch\E-first!";  
print "$str\n";
```

Backslash non alpha-numeric including spaces.

```
$str = "\QWelcome to ericsonbatch's family";  
print "$str\n";
```

Perl Datatypes

This will produce the following result –

Welcome to

ericsonbatch!

Welcome to \nericsonbatch!

Welcome to ericsonbatch!

WELCOME TO ERICSONBATCH!

Welcome to ERICSONBATCH-first!

Welcome\ to\ ericsonbatch\'s\ family

Perl - Scalars

Creating variables

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

```
print "Age = $age\n";  
print "Name = $name\n";  
print "Salary = $salary\n";
```

This will produce the following result –

```
Age = 25  
Name = John Paul  
Salary = 1445.5
```

Perl Variables – Numeric Scalars

```
$integer = 200;
```

```
$negative = -300;
```

```
$floating = 200.340;
```

```
$bigfloat = -1.2E-23;
```

```
# 377 octal, same as 255 decimal
```

```
$octal = 0377;
```

```
# FF hex, also 255 decimal
```

```
$hexa = 0xff;
```

```
print "integer = $integer\n";
```

```
print "negative = $negative\n";
```

```
print "floating = $floating\n";
```

```
print "bigfloat = $bigfloat\n";
```

```
print "octal = $octal\n";
```

```
print "hexa = $hexa\n";
```

This will produce the following result –

```
integer = 200
```

```
negative = -300
```

```
floating = 200.34
```

```
bigfloat = -1.2e-23
```

```
octal = 255
```

```
hexa = 255
```


Perl Variables – String Scalars

```
$var = "This is string scalar!";  
$quote = 'I m inside single quote - $var';  
$double = "This is inside single quote - $var";  
  
$escape = "This example of escape -\tHello, World!";  
  
print "var = $var\n";  
print "quote = $quote\n";  
print "double = $double\n";  
print "escape = $escape\n";
```

This will produce the following result –

```
var = This is string scalar!  
quote = I m inside single quote - $var  
double = This is inside single quote - This is string scalar!  
escape = This example of escape - Hello, World!
```

Perl Variables – Scalar operators

```
# Concatenates strings
$str = "hello" . "world";

# adds two numbers.
$num = 5 + 10;

# multiplies two numbers.
$mul = 4 * 5;

# concatenates string and number.
$mix = $str . $num;
```

```
print "str = $str\n";
print "num = $num\n";
Print "mul = $mul\n";
print "mix = $mix\n";
```

This will produce the following result –

```
str = helloworld
num = 15
mul = 20
mix = helloworld15
```

Perl Variables – Multiline strings

```
$string = 'This is  
a multiline  
string';
```

```
print "$string\n";
```

This will produce the following result –

```
This is  
a multiline  
string
```

```
print <<EOF;
```

```
This is  
a multiline  
string  
EOF
```

This will also produce the same result –

```
This is  
a multiline  
string
```

Perl Variables – Special literals

```
print "File name ". __FILE__ . "\n";  
print "Line Number " . __LINE__ . "\n";  
print "Package " . __PACKAGE__ . "\n";
```

they can not be interpolated

```
print "__FILE__ __LINE__ __PACKAGE__\n";
```

This will produce the following result –

File name main.pl

Line Number 4

Package main

```
__FILE__ __LINE__ __PACKAGE__
```

Perl Variables – Arrays

```
@ages = (25, 30, 40);  
@names = ("John Paul", "Lisa", "Kumar");
```

```
print "\$ages[0] = $ages[0]\n";  
print "\$ages[1] = $ages[1]\n";  
print "\$ages[2] = $ages[2]\n";  
print "\$names[0] = $names[0]\n";  
print "\$names[1] = $names[1]\n";  
print "\$names[2] = $names[2]\n";
```

This will produce the following result –

```
$ages[0] = 25  
$ages[1] = 30  
$ages[2] = 40  
$names[0] = John Paul  
$names[1] = Lisa  
$names[2] = Kumar
```

Perl Variables – Array creation

```
@array = (1, 2, 'Hello');
```

```
@array = qw/This is an array/;
```

```
@days = qw/Monday
```

```
Tuesday
```

```
Wednesday
```

```
Thursday
```

```
Friday
```

```
Saturday
```

```
Sunday/;
```

```
@days = qw/Mon Tue Wed Thu Fri Sat Sun/;
```

```
print "$days[0]\n";
```

```
print "$days[1]\n";
```

```
print "$days[2]\n";
```

```
print "$days[6]\n";
```

```
print "$days[-1]\n";
```

```
print "$days[-7]\n";
```

This will produce the following result –

Mon

Tue

Wed

Sun

Sun

Mon

Perl Variables – Sequential arrays

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

```
@var_10 = (1..10);
```

```
@var_20 = (10..20);
```

```
@var_abc = (a..z);
```

```
print "@var_10\n"; # Prints number from 1 to 10
```

```
print "@var_20\n"; # Prints number from 10 to 20
```

```
print "@var_abc\n"; # Prints number from a to z
```

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

```
10 11 12 13 14 15 16 17 18 19 20
```

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Perl Variables – Array size

```
@array = (1,2,3);  
print "Size: ",scalar @array,"\n";
```

```
@array = (1,2,3);  
$array[50] = 4;
```

```
$size = @array;  
$max_index = $#array;
```

```
print "Size: $size\n";  
print "Max Index: $max_index\n";
```

This will produce the following result –

Size: 51

Max Index: 50

Perl Variables – Adding & removing elements

create a simple array

```
@coins = ("Quarter","Dime","Nickel");  
print "1. \@coins = @coins\n";
```

add one element at the end of the array

```
push(@coins, "Penny");  
print "2. \@coins = @coins\n";
```

add one element at the beginning of the array

```
unshift(@coins, "Dollar");  
print "3. \@coins = @coins\n";
```

1. @coins = Quarter Dime Nickel

2. @coins = Quarter Dime Nickel Penny

3. @coins = Dollar Quarter Dime Nickel Penny

Perl Variables – Adding & removing elements

remove one element from the last of the array.

```
pop(@coins);
```

```
print "4. \@coins = @coins\n";
```

remove one element from the beginning of the array.

```
shift(@coins);
```

```
print "5. \@coins = @coins\n";
```

4. @coins = Dollar Quarter Dime Nickel

5. @coins = Quarter Dime Nickel

Perl Variables – Slicing array elements

```
@days = qw/Mon Tue Wed Thu Fri Sat Sun/;
```

```
@weekdays = @days[3,4,5];
```

```
print "@weekdays\n";
```

This will produce the following result –

Thu Fri Sat

```
@days = qw/Mon Tue Wed Thu Fri Sat Sun/;
```

```
@weekdays = @days[2..6];
```

```
print "@weekdays\n";
```

This will produce the following result –

Wed Thu Fri Sat Sun

Perl Variables – Replacing array elements

```
splice @ARRAY, OFFSET [ , LENGTH [ , LIST ] ]
```

```
@nums = (1..20);  
print "Before - @nums\n";
```

```
splice(@nums, 5, 5, 21..25);  
print "After - @nums\n";
```

This will produce the following result –

```
Before - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

```
After - 1 2 3 4 5 21 22 23 24 25 11 12 13 14 15 16 17 18 19 20
```

Perl Variables – String to arrays

```
# define Strings
$var_string = "Rain-Drops-On-Roses-And-Whiskers-On-Kittens";
$var_names = "Larry,David,Roger,Ken,Michael,Tom";

# transform above strings into arrays.
@string = split('-', $var_string);
@names = split(',', $var_names);
print "$string[3]\n"; # This will print Roses
print "$names[4]\n"; # This will print Michael
```

This will produce the following result –

Roses

Michael

Perl Variables – Array to string

```
# define Strings
$var_string = "Rain-Drops-On-Roses-And-Whiskers-On-Kittens";
$var_names = "Larry,David,Roger,Ken,Michael,Tom";
# transform above strings into arrays.
@string = split('-', $var_string);
@names = split(',', $var_names);
$string1 = join( '-', @string );
$string2 = join( ',', @names );

print "$string1\n";
print "$string2\n";
```

This will produce the following result –

```
Rain-Drops-On-Roses-And-Whiskers-On-Kittens
Larry,David,Roger,Ken,Michael,Tom
```

Perl Variables – \$[Special Variable

```
# define an array
```

```
@foods = qw(pizza steak chicken burgers);
```

```
print "Foods: @foods\n";
```

```
# Let's reset first index of all the arrays.
```

```
$[ = 1;
```

```
print "Food at \@foods[1]: $foods[1]\n";
```

```
print "Food at \@foods[2]: $foods[2]\n";
```

This will produce the following result–

Foods: pizza steak chicken burgers

Food at @foods[1]: pizza

Food at @foods[2]: steak

Perl Variables – Merging arrays

```
@numbers = (1,3,(4,5,6));
```

```
print "numbers = @numbers\n";
```

This will produce the following result –

```
numbers = 1 3 4 5 6
```

```
@odd = (1,3,5);
```

```
@even = (2, 4, 6);
```

```
@numbers = (@odd, @even);
```

```
print "numbers = @numbers\n";
```

This will produce the following result –

```
numbers = 1 3 5 2 4 6
```


Perl Variables – Select from lists

```
$var = (5,4,3,2,1)[4];
```

```
print "value of var = $var\n"
```

This will produce the following result –

```
value of var = 1
```

```
@list = (5,4,3,2,1)[1..3];
```

```
print "Value of list = @list\n";
```

This will produce the following result –

```
Value of list = 4 3 2
```

Perl Variables – Hashes

```
%data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);  
  
print "\$data{'John Paul'} = $data{'John Paul'}\n";  
print "\$data{'Lisa'} = $data{'Lisa'}\n";  
print "\$data{'Kumar'} = $data{'Kumar'}\n";
```

This will produce the following result –

```
$data{'John Paul'} = 45  
$data{'Lisa'} = 30  
$data{'Kumar'} = 40
```

Perl Variables – Creating Hashes

```
$data{'John Paul'} = 45;
```

```
$data{'Lisa'} = 30;
```

```
$data{'Kumar'} = 40;
```

```
%data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);
```

```
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
```

```
%data = (-JohnPaul => 45, -Lisa => 30, -Kumar => 40); # you cannot use spaces in the words in this method
```

```
$val = %data{-JohnPaul}
```

```
$val = %data{-Lisa}
```

Perl Variables – Accessing elements

```
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
```

```
print "$data{'John Paul'}\n";
```

```
print "$data{'Lisa'}\n";
```

```
print "$data{'Kumar'}\n";
```

This will produce the following result –

45

30

40

Perl Variables – Extracting slices

```
%data = (-JohnPaul => 45, -Lisa => 30, -Kumar => 40);
```

```
@array = @data{-JohnPaul, -Lisa};
```

```
print "Array : @array\n";
```

This will produce the following result –

```
Array : 45 30
```

Perl Variables – Extracting keys & values

```
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
```

```
@names = keys %data;
```

```
print "$names[0]\n";
```

```
print "$names[1]\n";
```

```
print "$names[2]\n";
```

This will produce the following result –

Lisa

John Paul

Kumar

Perl Variables – Extracting keys & values

```
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
```

```
@ages = values %data;
```

```
print "$ages[0]\n";
```

```
print "$ages[1]\n";
```

```
print "$ages[2]\n";
```

This will produce the following result –

30

45

40

Perl Variables – Checking if exists

```
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
```

```
if( exists($data{'Lisa'}) ){  
    print "Lisa is $data{'Lisa'} years old\n";  
}  
else{  
    print "I don't know age of Lisa\n";  
}
```

Result:

Lisa is 30 years old

Perl Variables – Checking Hash size

```
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
```

```
@keys = keys %data;
```

```
$size = @keys;
```

```
print "1 - Hash size: is $size\n";
```

```
@values = values %data;
```

```
$size = @values;
```

```
print "2 - Hash size: is $size\n";
```

This will produce the following result –

1 - Hash size: is 3

2 - Hash size: is 3

Perl Variables – Add & remove elements

```
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);  
@keys = keys %data;  
$size = @keys;  
print "1 - Hash size: is $size\n";
```

```
# adding an element to the hash;  
$data{'Ali'} = 55;  
@keys = keys %data;  
$size = @keys;  
print "2 - Hash size: is $size\n";
```

```
# delete the same element from the hash;  
delete $data{'Ali'};  
@keys = keys %data;  
$size = @keys;  
print "3 - Hash size: is $size\n";
```

This will produce the following result –

```
1 - Hash size: is 3  
2 - Hash size: is 4  
3 - Hash size: is 3
```

Perl – Conditional statements

if statement

An if statement consists of a boolean expression followed by one or more statements.

if...else statement

An if statement can be followed by an optional else statement.

if...elsif...else statement

An if statement can be followed by an optional elsif statement and then by an optional else statement.

Perl – Conditional statements

unless statement

An unless statement consists of a boolean expression followed by one or more statements.

unless...else statement

An unless statement can be followed by an optional else statement.

unless...elsif..else statement

An unless statement can be followed by an optional elsif statement and then by an optional else statement.

switch statement

With 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.

Perl – if statement

```
$a = 10;  
# check the boolean condition using if statement  
if( $a < 20 ){  
    # if condition is true then print the following  
    printf "a is less than 20\n";  
}  
print "value of a is : $a\n";
```

```
$a = "";  
# check the boolean condition using if statement  
if( $a ){  
    # if condition is true then print the following  
    printf "a has a true value\n";  
}  
print "value of a is : $a\n";
```

Result:

a is less than 20

value of a is : 10

value of a is :

Perl – if...else statements

```
$a = 100;
# check the boolean condition using if statement
if( $a < 20 ){
    # if condition is true then print the following
    printf "a is less than 20\n";
}else{
    # if condition is false then print the following
    printf "a is greater than 20\n";
}
print "value of a is : $a\n";
```

Result:

a is greater than 20

value of a is : 100

```
$a = "";
# check the boolean condition using if statement
if( $a ){
    # if condition is true then print the following
    printf "a has a true value\n";
}else{
    # if condition is false then print the following
    printf "a has a false value\n";
}
print "value of a is : $a\n";
```

Result:

a has a false value

value of a is :

Perl – if...elseif statements

```
$a = 100;
# check the boolean condition using if statement
if( $a == 20 ){
    # if condition is true then print the following
    printf "a has a value which is 20\n";
}elseif( $a == 30 ){
    # if condition is true then print the following
    printf "a has a value which is 30\n";
}else{
    # if none of the above conditions is true
    printf "a has a value which is $a\n";
}
```

Result:

a has a value which is 100

Perl – unless statements

```
$a = 20;  
# check the boolean condition using unless statement  
unless( $a < 20 ){  
    # if condition is false then print the following  
    printf "a is not less than 20\n";  
}  
print "value of a is : $a\n";
```

```
$a = "";  
# check the boolean condition using unless statement  
unless ( $a ){  
    # if condition is false then print the following  
    printf "a has a false value\n";  
}  
print "value of a is : $a\n";
```

Result:

a is not less than 20

value of a is : 20

a has a false value

value of a is :

Perl – unless...else statements

```
$a = 100;  
# check the boolean condition using unless statement  
unless( $a == 20 ){  
    # if condition is false then print the following  
    printf "given condition is false\n";  
}else{  
    # if condition is true then print the following  
    printf "given condition is true\n";  
}  
print "value of a is : $a\n";
```

Result:

given condition is false
value of a is : 100

```
$a = "";  
# check the boolean condition using unless statement  
unless( $a ){  
    # if condition is false then print the following  
    printf "a has a false value\n";  
}else{  
    # if condition is true then print the following  
    printf "a has a true value\n";  
}  
print "value of a is : $a\n";
```

Result:

a has a false value
value of a is :

Perl – unless...elseif statements

```
$a = 20;  
# check the boolean condition using if statement  
unless( $a == 30 ){  
    # if condition is false then print the following  
    printf "a has a value which is not 20\n";  
}elseif( $a == 30 ){  
    # if condition is true then print the following  
    printf "a has a value which is 30\n";  
}else{  
    # if none of the above conditions is met  
    printf "a has a value which is $a\n";  
}
```

Result:

a has a value which is not 20

Perl – switch statements

```
use Switch;
```

Result:

```
$var = 10;
```

number 100

```
@array = (10, 20, 30);
```

```
%hash = ('key1' => 10, 'key2' => 20);
```

```
switch($var){
```

```
    case 10      { print "number 100\n" }
```

```
    case "a"     { print "string a" }
```

```
    case [1..10,42] { print "number in list" }
```

```
    case (\@array) { print "number in list" }
```

```
    case (\%hash)  { print "entry in hash" }
```

```
    else         { print "previous case not true" }
```

```
}
```

Perl – switch statements

```
use Switch;
```

Result:

```
$var = 10;
```

number 100

```
@array = (10, 20, 30);
```

number in list

```
%hash = ('key1' => 10, 'key2' => 20);
```

```
switch($var){
```

```
    case 10      { print "number 100\n"; next; }
```

```
    case "a"     { print "string a" }
```

```
    case [1..10,42] { print "number in list" }
```

```
    case (\@array) { print "number in list" }
```

```
    case (\%hash)  { print "entry in hash" }
```

```
    else         { print "previous case not true" }
```

```
}
```

Perl – ?: operator

```
$name = "Ali";
```

```
$age = 10;
```

```
$status = ($age > 60 )? "A senior citizen" : "Not a senior citizen";
```

```
print "$name is - $status\n";
```

This will produce the following result –

Ali is - Not a senior citizen

Perl – Loops

while loop

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

until loop

Repeats a statement or group of statements until a given condition becomes true. It tests the condition before executing the loop body.

for loop

Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

Perl – Loops

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.

do...while loop

Like a while statement, except that it tests the condition at the end of the loop body

nested loops

You can use one or more loop inside any another while, for or do..while loop.

Perl – while loop

```
$a = 10;
```

```
# while loop execution
```

```
while( $a < 20 ){
```

```
    printf "Value of a: $a\n";
```

```
    $a = $a + 1;
```

```
}
```

Result:

Value of a: 10

Value of a: 11

Value of a: 12

Value of a: 13

Value of a: 14

Value of a: 15

Value of a: 16

Value of a: 17

Value of a: 18

Value of a: 19

Perl – until loop

```
$a = 5;
```

```
# until loop execution
```

```
until( $a > 10 ){  
    printf "Value of a: $a\n";  
    $a = $a + 1;  
}
```

Result:

Value of a: 5

Value of a: 6

Value of a: 7

Value of a: 8

Value of a: 9

Value of a: 10

Perl – for loop

```
# for loop execution
for( $a = 10; $a < 20; $a = $a + 1 ){
    print "value of a: $a\n";
}
```

Result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Perl – foreach loop

```
@list = (2, 20, 30, 40, 50);
```

```
# foreach loop execution  
foreach $a (@list){  
    print "value of a: $a\n";  
}
```

Result:

```
value of a: 2  
value of a: 20  
value of a: 30  
value of a: 40  
value of a: 50
```

Perl – do...while loop

```
$a = 10;
```

```
# do...while loop execution
```

```
do{
```

```
    printf "Value of a: $a\n";
```

```
    $a = $a + 1;
```

```
}while( $a < 20 );
```

Result:

Value of a: 10

Value of a: 11

Value of a: 12

Value of a: 13

Value of a: 14

Value of a: 15

Value of a: 16

Value of a: 17

Value of a: 18

Value of a: 19

Perl – nested loop

```
$a = 0;
$b = 0;
# outer while loop
while($a < 3){
    $b = 0;
    # inner while loop
    while( $b < 3 ){
        print "value of a = $a, b = $b\n";
        $b = $b + 1;
    }
    $a = $a + 1;
    print "Value of a = $a\n\n";
}
```

value of a = 0, b = 0

value of a = 0, b = 1

value of a = 0, b = 2

Value of a = 1

value of a = 1, b = 0

value of a = 1, b = 1

value of a = 1, b = 2

Value of a = 2

value of a = 2, b = 0

value of a = 2, b = 1

value of a = 2, b = 2

Value of a = 3

Perl – loop control

next statement

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

last statement

Terminates the loop statement and transfers execution to the statement immediately following the loop.

continue statement

A continue BLOCK, it is always executed just before the conditional is about to be evaluated again.

Perl – loop control

redo statement

The redo command restarts the loop block without evaluating the conditional again. The continue block, if any, is not executed.

goto statement

Perl supports a goto command with three forms: goto label, goto expr, and goto &name.

Perl – loop – next statement

<code>\$a = 10;</code>	value of a: 10
<code>while(\$a < 20){</code>	value of a: 11
<code>if(\$a == 15)</code>	value of a: 12
<code>{</code>	value of a: 13
<code># skip the iteration.</code>	value of a: 14
<code>\$a = \$a + 1;</code>	value of a: 16
<code>next;</code>	value of a: 17
<code>}</code>	value of a: 18
<code>print "value of a: \$a\n";</code>	value of a: 19
<code>\$a = \$a + 1;</code>	
<code>}</code>	

Perl – loop – next statement

Let's take one example where we are going to use a LABEL along with next statement –

```
$a = 0;
```

```
OUTER: while( $a < 4 ){
```

```
    $b = 0;
```

```
    print "value of a: $a\n";
```

```
    INNER:while ( $b < 4){
```

```
        if( $a == 2){
```

```
            $a = $a + 1;
```

```
            # jump to outer loop
```

```
            next OUTER;
```

```
        }
```

```
        $b = $b + 1;
```

```
        print "Value of b : $b\n";
```

```
    }
```

```
    print "\n";
```

```
    $a = $a + 1;
```

```
}
```

Perl – loop – last statement

```
$a = 10;
while( $a < 20 ){
    if( $a == 15)
    {
        # terminate the loop.
        $a = $a + 1;
        last;
    }
    print "value of a: $a\n";
    $a = $a + 1;
}
```

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

Perl – loop – last statement

```
$a = 0;
OUTER: while( $a < 4 ){
    $b = 0;
    print "value of a: $a\n";
    INNER:while ( $b < 4){
        if( $a == 2){
            # terminate outer loop
            last OUTER;
        }
        $b = $b + 1;
        print "Value of b : $b\n";
    }
    print "\n";
    $a = $a + 1;
}
```

Perl – loop – continue statement

```
$a = 0;
while($a < 3){
    print "Value of a = $a\n";
}continue{
    $a = $a + 1;
}
```

This would produce the following result –

```
Value of a = 0
Value of a = 1
Value of a = 2
```

```
@list = (1, 2, 3, 4, 5);
foreach $a (@list){
    print "Value of a = $a\n";
}continue{
    last if $a == 4;
}
```

This would produce the following result –

```
Value of a = 1
Value of a = 2
Value of a = 3
Value of a = 4
```

Perl – loop – redo statement

```
$a = 0;
while($a < 10){
    if( $a == 5 ){
        $a = $a + 1;
        redo;
    }
    print "Value of a = $a\n";
}continue{
    $a = $a + 1;
}
```

Value of a = 0

Value of a = 1

Value of a = 2

Value of a = 3

Value of a = 4

Value of a = 5

Value of a = 6

Value of a = 7

Value of a = 8

Value of a = 9

Perl – loop – goto statement

`goto LABEL`

The `goto LABEL` form jumps to the statement labeled with `LABEL` and resumes execution from there.

`goto EXPR`

The `goto EXPR` form is just a generalization of `goto LABEL`. It expects the expression to return a label name and then jumps to that labeled statement.

`goto &NAME`

It substitutes a call to the named subroutine for the currently running subroutine.

Perl – loop – goto LABEL statement

```
$a = 10;
```

Value of a = 10

```
LOOP:do
```

Value of a = 11

```
{
```

Value of a = 12

```
  if( $a == 15){
```

Value of a = 13

```
    # skip the iteration.
```

Value of a = 14

```
    $a = $a + 1;
```

Value of a = 16

```
    # use goto LABEL form
```

Value of a = 17

```
    goto LOOP;
```

Value of a = 18

```
}
```

Value of a = 19

```
  print "Value of a = $a\n";
```

```
  $a = $a + 1;
```

```
}while( $a < 20 );
```

Perl – loop – goto EXPR statement

<code>\$a = 10;</code>	Value of a = 10
<code>\$str1 = "LO";</code>	Value of a = 11
<code>\$str2 = "OP";</code>	Value of a = 12
<code>LOOP:do</code>	Value of a = 13
<code>{</code>	Value of a = 14
<code> if(\$a == 15){</code>	Value of a = 16
<code> # skip the iteration.</code>	Value of a = 17
<code> \$a = \$a + 1;</code>	Value of a = 18
<code> # use goto EXPR form</code>	Value of a = 19
<code> goto \$str1.\$str2;</code>	
<code> }</code>	
<code> print "Value of a = \$a\n";</code>	
<code> \$a = \$a + 1;</code>	
<code>}while(\$a < 20);</code>	

Perl – operators

Arithmetic Operators

Equality Operators

Logical Operators

Assignment Operators

Bitwise Operators

Logical Operators

Quote-like Operators

Miscellaneous Operators

Perl – arithmetic operator

Operator	Description	Example
+	Addition - Adds values on either side of the operator	\$a + \$b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	\$a - \$b will give -10
*	Multiplication - Multiplies values on either side of the operator	\$a * \$b will give 200
/	Division - Divides left hand operand by right hand operand	\$b / \$a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	\$b % \$a will give 0
**	Exponent - Performs exponential (power) calculation on operators	\$a**\$b will give 10 to the power 20

Perl – arithmetic operator

```
$a = 21;
```

```
$b = 10;
```

```
print "Value of \ $a = $a and value of \ $b = $b\n";
```

```
$c = $a + $b;
```

```
print 'Value of $a + $b = ' . $c . "\n";
```

```
$c = $a - $b;
```

```
print 'Value of $a - $b = ' . $c . "\n";
```

```
$c = $a * $b;
```

```
print 'Value of $a * $b = ' . $c . "\n";
```

```
$c = $a / $b;
```

```
print 'Value of $a / $b = ' . $c . "\n";
```

```
$c = $a % $b;
```

```
print 'Value of $a % $b = ' . $c . "\n";
```

```
$a = 2;
```

```
$b = 4;
```

```
$c = $a ** $b;
```

```
print 'Value of $a ** $b = ' . $c . "\n";
```

Perl – equality operator

Operator	Description	Example
<code>==</code>	Checks if the value of two operands are equal or not, if yes then condition becomes true.	<code>(\$a == \$b)</code> is not true.
<code>!=</code>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	<code>(\$a != \$b)</code> is true.
<code><=></code>	Checks if the value of two operands are equal or not, and returns -1, 0, or 1 depending on whether the left argument is numerically less than, equal to, or greater than the right argument.	<code>(\$a <=> \$b)</code> returns -1.
<code>></code>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	<code>(\$a > \$b)</code> is not true.
<code><</code>	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	<code>(\$a < \$b)</code> is true.
<code>>=</code>	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	<code>(\$a >= \$b)</code> is not true.
<code><=</code>	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	<code>(\$a <= \$b)</code> is true.

Perl – equality operator

```
$a = 21;
$b = 10;
print "Value of \$a = $a and value of \$b = $b\n";

if( $a == $b ){
    print "$a == \$b is true\n";
}else{
    print "\$a == \$b is not true\n";
}
```

```
if( $a != $b ){
    print "\$a != \$b is true\n";
}else{
    print "\$a != \$b is not true\n";
}
```

```
$c = $a <=> $b;
print "\$a <=> \$b returns $c\n";
```

```
if( $a > $b ){
    print "\$a > \$b is true\n";
}else{
    print "\$a > \$b is not true\n";
}
```

Perl – equality operator

```
if( $a >= $b ){  
    print "\$a >= \$b is true\n";  
}else{  
    print "\$a >= \$b is not true\n";  
}
```

```
if( $a < $b ){  
    print "\$a < \$b is true\n";  
}else{  
    print "\$a < \$b is not true\n";  
}
```

```
if( $a <= $b ){  
    print "\$a <= \$b is true\n";  
}else{  
    print "\$a <= \$b is not true\n";  
}
```

Value of \$a = 21 and value of \$b = 10

\$a == \$b is not true

\$a != \$b is true

\$a <=> \$b returns 1

\$a > \$b is true

\$a >= \$b is true

\$a < \$b is not true

\$a <= \$b is not true

Perl – equality operator

Operator	Description	Example
lt	Returns true if the left argument is stringwise less than the right argument.	(\$a lt \$b) is true.
gt	Returns true if the left argument is stringwise greater than the right argument.	(\$a gt \$b) is false.
le	Returns true if the left argument is stringwise less than or equal to the right argument.	(\$a le \$b) is true.
ge	Returns true if the left argument is stringwise greater than or equal to the right argument.	(\$a ge \$b) is false.
eq	Returns true if the left argument is stringwise equal to the right argument.	(\$a eq \$b) is false.
ne	Returns true if the left argument is stringwise not equal to the right argument.	(\$a ne \$b) is true.
cmp	Returns -1, 0, or 1 depending on whether the left argument is stringwise less than, equal to, or greater than the right argument.	(\$a cmp \$b) is -1.

Perl – equality operator

```
$a = "abc";  
$b = "xyz";  
print "Value of \$a = $a and value of \$b = $b\n";  
if( $a lt $b ){  
    print "$a lt \$b is true\n";  
}  
else{  
    print "\$a lt \$b is not true\n";  
}  
  
if( $a gt $b ){  
    print "\$a gt \$b is true\n";  
}  
else{  
    print "\$a gt \$b is not true\n";  
}
```

```
if( $a le $b ){  
    print "\$a le \$b is true\n";  
}  
else{  
    print "\$a le \$b is not true\n";  
}
```

```
if( $a ge $b ){  
    print "\$a ge \$b is true\n";  
}  
else{  
    print "\$a ge \$b is not true\n";  
}
```

```
if( $a ne $b ){  
    print "\$a ne \$b is true\n";  
}  
else{
```


Perl – equality operator

```
if( $a ne $b ){  
    print "\$a ne \$b is true\n";  
}else{  
    print "\$a ne \$b is not true\n";  
}  
  
$c = $a cmp $b;  
print "\$a cmp \$b returns $c\n";
```

When the above code is executed, it produces following results:

Value of \$a = abc and value of \$b = xyz

abc lt \$b is true

\$a gt \$b is not true

\$a le \$b is true

\$a ge \$b is not true

\$a ne \$b is true

\$a cmp \$b returns -1

Perl – assignment operator

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	\$c = \$a + \$b will assigned value of \$a + \$b into \$c
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	\$c += \$a is equivalent to \$c = \$c + \$a
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	\$c -= \$a is equivalent to \$c = \$c - \$a
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	\$c *= \$a is equivalent to \$c = \$c * \$a
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	\$c /= \$a is equivalent to \$c = \$c / \$a
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	\$c %= \$a is equivalent to \$c = \$c % a
**=	Exponent AND assignment operator, Performs exponential (power) calculation on operators and assign value to the left operand	\$c **= \$a is equivalent to \$c = \$c ** \$a

Perl – assignment operator

```
$a = 10;
```

```
$b = 20;
```

```
print "Value of \$a = $a and value of \$b = $b\n";
```

```
$c = $a + $b;
```

```
print "After assignment value of \$c = $c\n";
```

```
$c += $a;
```

```
print "Value of \$c = $c after statement \$c += \$a\n";
```

```
$c -= $a;
```

```
print "Value of \$c = $c after statement \$c -= \$a\n";
```

```
$c *= $a;
```

```
print "Value of \$c = $c after statement \$c *= \$a\n";
```

```
$c /= $a;
```

```
print "Value of \$c = $c after statement \$c /= \$a\n";
```

```
$c %= $a;
```

```
print "Value of \$c = $c after statement \$c %= \$a\n";
```

```
$c = 2;
```

```
$a = 4;
```

```
print "Value of \$a = $a and value of \$c = $c\n";
```

```
$c **= $a;
```

```
print "Value of \$c = $c after statement \$c **= \$a\n";
```

Perl – assignment operator

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

Value of \$a = 10 and value of \$b = 20

After assignment value of \$c = 30

Value of \$c = 40 after statement `$c += $a`

Value of \$c = 30 after statement `$c -= $a`

Value of \$c = 300 after statement `$c *= $a`

Value of \$c = 30 after statement `$c /= $a`

Value of \$c = 0 after statement `$c %= $a`

Value of \$a = 4 and value of \$c = 2

Value of \$c = 16 after statement `$c **= $a`

Perl – logical operator

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(\$a and \$b) is false.
&&	C-style Logical AND operator copies a bit to the result if it exists in both operands.	(\$a && \$b) is false.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(\$a or \$b) is true.
	C-style Logical OR operator copies a bit if it exists in eather operand.	(\$a \$b) is true.
not	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	not(\$a and \$b) is false.

Perl – logical operator

```
$a = true;  
$b = false;  
print "Value of \$a = $a and value of \$b = $b\n";  
$c = ($a and $b);  
print "Value of \$a and \$b = $c\n";
```

```
$c = ($a && $b);  
print "Value of \$a && \$b = $c\n";
```

```
$c = ($a or $b);  
print "Value of \$a or \$b = $c\n";
```

```
$c = ($a || $b);  
print "Value of \$a || \$b = $c\n";
```

```
$a = 0;  
$c = not($a);  
print "Value of not(\$a)= $c\n";
```

Result -

Value of \$a = true and value of \$b = false

Value of \$a and \$b = false

Value of \$a && \$b = false

Value of \$a or \$b = true

Value of \$a || \$b = true

Value of not(\$a)= 1

Perl – quote like operator

Operator	Description	Example
<code>q{ }</code>	Encloses a string with-in single quotes	<code>q{abcd}</code> gives <code>'abcd'</code>
<code>qq{ }</code>	Encloses a string with-in double quotes	<code>qq{abcd}</code> gives <code>"abcd"</code>
<code>qx{ }</code>	Encloses a string with-in invert quotes	<code>qx{abcd}</code> gives <code>`abcd`</code>

Perl – quote like operator

```
$a = 10;
```

```
$b = q{a = $a};
```

```
print "Value of q{a = \"$a\"} = $b\n";
```

```
$b = qq{a = $a};
```

```
print "Value of qq{a = \"$a\"} = $b\n";
```

```
# unix command execution
```

```
$t = qx{date};
```

```
print "Value of qx{date} = $t\n";
```

Result –

Value of q{a = \$a} = a = \$a

Value of qq{a = \$a} = a = 10

Value of qx{date} = Thu Feb 14 08:13:17 MST 2013

Perl – miscellaneous operator

Operator	Description	Example
.	Binary operator dot (.) concatenates two strings.	If \$a="abc", \$b="def" then \$a.\$b will give "abcdef"
x	The repetition operator x returns a string consisting of the left operand repeated the number of times specified by the right operand.	('-' x 3) will give ---.
..	The range operator .. returns a list of values counting (up by ones) from the left value to the right value	(2..5) will give (2, 3, 4, 5)
++	Auto Increment operator increases integer value by one	\$a++ will give 11
--	Auto Decrement operator decreases integer value by one	\$a-- will give 9
->	The arrow operator is mostly used in dereferencing a method or variable from an object or a class name	\$obj->\$a is an example to access variable \$a from object \$obj.

Perl – miscellaneous operator

```
$a = "abc";  
$b = "def";  
print "Value of \$a = $a and value of \$b = $b\n";
```

```
$c = $a . $b;  
print "Value of \$a . \$b = $c\n";
```

```
$c = "-" x 3;  
print "Value of \"-\" x 3 = $c\n";
```

```
@c = (2..5);  
print "Value of (2..5) = @c\n";
```

```
$a = 10;  
$b = 15;  
print "Value of \$a = $a and value of \$b = $b\n";
```

```
$a++;  
$c = $a ;  
print "Value of \$a after \$a++ = $c\n";
```

```
$b--;  
$c = $b ;  
print "Value of \$b after \$b-- = $c\n";
```

Perl – miscellaneous operator

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

Value of \$a = abc and value of \$b = def

Value of \$a . \$b = abcdef

Value of "-" x 3 = ---

Value of (2..5) = 2 3 4 5

Value of \$a = 10 and value of \$b = 15

Value of \$a after \$a++ = 11

Value of \$b after \$b-- = 14

Perl – Subroutines

A Perl subroutine or function is a group of statements that together performs a task. You can divide up your code into separate subroutines.

The general form of a subroutine definition in Perl programming language is as follows –

```
sub subroutine_name{  
    body of the subroutine  
}
```

The typical way of calling that Perl subroutine is as follows –

```
subroutine_name( list of arguments );
```

Perl – Subroutines

Function definition

```
sub Hello{  
    print "Hello, World!\n";  
}
```

Function call

```
Hello();
```

When above program is executed, it produces the following result –

Hello, World!

Perl – Subroutines – passing arguments

Function definition

```
sub Average{
```

```
    # get total number of arguments passed.
```

```
    $n = scalar(@_);
```

```
    $sum = 0;
```

```
    foreach $item (@_){
```

```
        $sum += $item;
```

```
    }
```

```
    $average = $sum / $n;
```

```
    print "Average for the given numbers : $average\n";
```

```
}
```

Function call

```
Average(10, 20, 30);
```

When above program is executed,
it produces the following result –

Average for the given numbers : 20

Perl – Subroutines – passing arguments

Passing Lists to Subroutines

Function definition

```
sub PrintList{  
    my @list = @_;  
    print "Given list is @list\n";  
}  
  
$a = 10;  
@b = (1, 2, 3, 4);
```

Function call with list parameter

```
PrintList($a, @b);
```

When above program is executed,
it produces the following result –

Given list is 10 1 2 3 4

Perl – Subroutines – passing arguments

Passing Hashes to Subroutines

Function definition

```
sub PrintHash{
```

```
    my (%hash) = @_;
```

```
    foreach my $key ( keys %hash ){
```

```
        my $value = $hash{$key};
```

```
        print "$key : $value\n";
```

```
    }
```

```
}
```

```
%hash = ('name' => 'Tom', 'age' => 19);
```

Function call with hash parameter

```
PrintHash(%hash);
```

When above program is executed,
it produces the following result –

```
name : Tom
```

```
age : 19
```


Perl – Subroutines – returning values

Function definition

```
sub Average{  
    # get total number of arguments passed.  
    $n = scalar(@_);  
    $sum = 0;  
  
    foreach $item (@_){  
        $sum += $item;  
    }  
    $average = $sum / $n;  
  
    return $average;  
}
```

Function call

```
$num = Average(10, 20, 30);  
print "Average for the given numbers : $num\n";
```

When above program is executed,
it produces the following result –

Average for the given numbers : 20

Perl – Subroutines – private variables - my

```
sub somefunc {  
    my $variable; # $variable is invisible outside somefunc()  
    my ($another, @an_array, %a_hash); # declaring many variables at once  
}
```

Global variable

```
$string = "Hello, World!";
```

Function definition

```
sub PrintHello{  
    # Private variable for PrintHello function  
    my $string;  
    $string = "Hello, Perl!";  
    print "Inside the function $string\n";  
}
```

Function call

```
PrintHello();  
print "Outside the function $string\n";
```

When above program is executed,
it produces the following result –

Inside the function Hello, Perl!
Outside the function Hello, World!

Perl – Subroutines – temp variables – local()

Global variable

```
$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";
```

When above program is executed,
it produces the following result –

Inside the function PrintMe Hello, Perl!

Inside the function PrintHello Hello, Perl!

Outside the function Hello, World!

Perl – Subroutines – state variable – state()

```
use feature 'state';
```

```
sub PrintCount{  
    state $count = 0; # initial value  
  
    print "Value of counter is $count\n";  
    $count++;  
}
```

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

When above program is executed,
it produces the following result –

```
Value of counter is 0  
Value of counter is 1  
Value of counter is 2  
Value of counter is 3  
Value of counter is 4
```

Perl – References

A Perl reference is a scalar data type that holds the location of another value which could be scalar, arrays, or hashes.

```
$scalarref = \ $foo;
```

```
$arrayref = \@ARGV;
```

```
$hashref = \%ENV;
```

```
$coderef = \&handler;
```

```
$globref = \*foo;
```

```
$arrayref = [1, 2, ['a', 'b', 'c']];
```

```
$hashref = {
```

```
    'Adam' => 'Eve',
```

```
    'Clyde' => 'Bonnie',
```

```
};
```

```
$coderef = sub { print "Boink!\n" };
```

Perl – Dereference

Dereferencing returns the value from a reference point to the location. To dereference a reference simply use \$, @ or % as prefix of the reference variable.

```
$var = 10;
# Now $r has reference to $var scalar.
$r = \ $var;
# Print value available at the location stored in $r.
print "Value of $var is : ", $$r, "\n";
@var = (1, 2, 3);
# Now $r has reference to @var array.
$r = \ @var;
# Print values available at the location stored in $r.
print "Value of @var is : ", @$r, "\n";
```

```
%var = ('key1' => 10, 'key2' => 20);
# Now $r has reference to %var hash.
$r = \ %var;
# Print values available at the location stored in $r.
print "Value of %var is : ", %$r, "\n";
```

Result:-

Value of 10 is : 10

Value of 1 2 3 is : 123

Value of %var is : key220key110

Perl – Dereference

```
$var = 10;  
$r = \ $var;  
print "Reference type in r : ", ref($r), "\n";
```

```
@var = (1, 2, 3);  
$r = \ @var;  
print "Reference type in r : ", ref($r), "\n";
```

```
%var = ('key1' => 10, 'key2' => 20);  
$r = \ %var;  
print "Reference type in r : ", ref($r), "\n";
```

Result:-

Reference type in r : SCALAR

Reference type in r : ARRAY

Reference type in r : HASH

Perl – Circular reference

```
my $foo = 100;
```

```
$foo = \ $foo;
```

```
print "Value of foo is : ", $$foo, "\n";
```

Result:-

Value of foo is : REF(0x9aae38)

Perl – References to functions

```
# Function definition
```

```
sub PrintHash{
```

```
    my (%hash) = @_;
```

```
    foreach $item (%hash){
```

```
        print "Item : $item\n";
```

```
    }
```

```
}
```

```
%hash = ('name' => 'Tom', 'age' => 19);
```

```
# Create a reference to above function.
```

```
$cref = \&PrintHash;
```

```
# Function call using reference.
```

```
&$cref(%hash);
```

When above program is executed, it produces the following result –

```
Item : name
```

```
Item : Tom
```

```
Item : age
```

```
Item : 19
```

Perl – OOP

Depends on:

References

Arrays

Hashes

Perl – OOP - Object

An object within Perl is a reference to a data type that knows what class it belongs to.

The object is stored as a reference in a scalar variable.

A scalar only contains a reference to the object, the same scalar can hold different objects in different classes.

Perl – OOP – Class

A class is a package that contains the corresponding methods required to create and manipulate objects.

Perl – OOP – bless()

To return the reference of the object. Ultimately becomes the object itself.

Syntax

Following is the simple syntax for this function –

```
bless REF, CLASSNAME
```

```
bless REF
```

Perl – OOP – Define class

To declare a class named Person in Perl we do –

```
package Person;
```

The scope of the package definition extends to the end of the file, or until another package keyword is encountered.

Perl – OOP – Objects creation and usage

The object constructor

```
package Person;
sub new
{
    my $class = shift;
    my $self = {
        _firstName => shift,
        _lastName  => shift,
        _ssn       => shift,
    };
};
```

```
# Print all the values just for clarification.
    print "First Name is $self->{_firstName}\n";
    print "Last Name is $self->{_lastName}\n";
    print "SSN is $self->{_ssn}\n";
    bless $self, $class;
    return $self;
}
```

Perl – OOP – Objects creation and usage

Creating the object:

```
$object = new Person( "Mohammad", "Saleem", 23234345);
```


Perl – OOP – Objects creation and usage

You can use simple hash in your constructor if you don't want to assign any value to any class variable. For example –

```
package Person;
sub new
{
    my $class = shift;
    my $self = {};
    bless $self, $class;
    return $self;
}
```

Perl – OOP – Define methods

For e.g. to get the person's first name –

```
sub getFirstName {  
    return $self->{_firstName};  
}
```

To set person's first name –

```
sub setFirstName {  
    my ( $self, $firstName ) = @_;  
    $self->{_firstName} = $firstName if defined($firstName);  
    return $self->{_firstName};  
}
```

Perl – OOP – Putting the first .pm file

```
package Person;
```

```
sub new
```

```
{
```

```
    my $class = shift;
```

```
    my $self = {
```

```
        _firstName => shift,
```

```
        _lastName  => shift,
```

```
        _ssn       => shift,
```

```
    };
```

```
    # Print all the values just for clarification.
```

```
        print "First Name is $self->{_firstName}\n";
```

```
        print "Last Name is $self->{_lastName}\n";
```

```
        print "SSN is $self->{_ssn}\n";
```

```
        bless $self, $class;
```

```
        return $self;
```

```
    }
```

Perl – OOP – Putting the first .pm file

```
sub setFirstName {  
    my ( $self, $firstName ) = @_;  
    $self->{_firstName} = $firstName if defined($firstName);  
    return $self->{_firstName};  
}
```

```
sub getFirstName {  
    my( $self ) = @_;  
    return $self->{_firstName};  
}  
1;
```

Perl – OOP – Calling the object & execute

```
use Person;
```

```
$object = new Person( "Mohammad", "Saleem", 23234345);
```

```
# Get first name which is set using constructor.
```

```
$firstName = $object->getFirstName();
```

```
print "Before Setting First Name is : $firstName\n";
```

```
$object->setFirstName( "Mohd." );
```

```
$firstName = $object->getFirstName();
```

```
print "Before Setting First Name is : $firstName\n";
```

Perl – OOP – Calling the object & execute

First Name is Mohammad

Last Name is Saleem

SSN is 23234345

Before Setting First Name is : Mohammad

Before Setting First Name is : Mohd.

Perl – OOP – Inheritance

Perl searches the class of the specified object for the given method or attribute, i.e., variable.

Perl searches the classes defined in the object class's @ISA array.

If no method is found in steps 1 or 2, then Perl uses an AUTOLOAD subroutine, if one is found in the @ISA tree.

If a matching method still cannot be found, then Perl searches for the method within the UNIVERSAL class (package) that comes as part of the standard Perl library.

If the method still has not found, then Perl gives up and raises a runtime exception.

Perl – OOP – Inheritance

```
package Employee;  
use Person;  
use strict;  
our @ISA = qw(Person); # inherits from Person
```


Perl – OOP – Inheritance

```
use Employee;
```

```
$object = new Employee( "Mohammad", "Saleem", 23234345);
```

```
# Get first name which is set using constructor.
```

```
$firstName = $object->getFirstName();
```

```
print "Before Setting First Name is : $firstName\n";
```

```
$object->setFirstName( "Mohd." );
```

```
$firstName = $object->getFirstName();
```

```
print "After Setting First Name is : $firstName\n";
```

Perl – OOP – Method overriding

The child class Employee inherits all the methods from the parent class Person.

To override those methods in your child class, you can do it by giving your own implementation.

You can add your additional functions in child class or you can add or modify the functionality of an existing methods in its parent class.

Perl – OOP – Method overriding

```
package Employee;
use Person;
use strict;
our @ISA = qw(Person); # inherits from Person

# Override constructor
sub new {
    my ($class) = @_;

    # Call the constructor of the parent class, Person.
    my $self = $class->SUPER::new( $_[1], $_[2], $_[3] );
    # Add few more attributes
    $self->{_id} = undef;
    $self->{_title} = undef;
```

```
    bless $self, $class;
    return $self;
}

# Override helper function
sub getFirstName {
    my( $self ) = @_;
    # This is child class function.
    print "This is child class helper function\n";
    return $self->{_firstName};
}
```

Perl – OOP – Method overriding

```
# Add more methods
sub setLastName{
    my ( $self, $lastName ) = @_;
    $self->{_lastName} = $lastName if defined($lastName);
    return $self->{_lastName};
}

sub getLastName {
    my( $self ) = @_;
    return $self->{_lastName};
}

1;
```

Perl – OOP – Method overriding

```
use Employee;
```

```
$object = new Employee( "Mohammad", "Saleem", 23234345);
```

```
# Get first name which is set using constructor.
```

```
$firstName = $object->getFirstName();
```

```
print "Before Setting First Name is : $firstName\n";
```

```
$object->setFirstName( "Mohd." );
```

```
$firstName = $object->getFirstName();
```

```
print "After Setting First Name is : $firstName\n";
```

Perl – OOP – Method overriding

First Name is Mohammad

Last Name is Saleem

SSN is 23234345

This is child class helper function

Before Setting First Name is : Mohammad

This is child class helper function

After Setting First Name is : Mohd.

Perl – OOP – AUTOLOAD

A default subroutine

By defining a function called `AUTOLOAD()`, any calls to undefined subroutines will call `AUTOLOAD()` function automatically.

The name of the missing subroutine is accessible within this subroutine as `$AUTOLOAD`.

Perl – OOP – AUTOLOAD

```
sub AUTOLOAD
{
    my $self = shift;
    my $type = ref ($self) || croak "$self is not an object";
    my $field = $AUTOLOAD;
    $field =~ s/.*://;
    unless (exists $self->{$field})
    {
        croak "$field does not exist in object/class $type";
    }
    if (@_)
    {
        return $self->($name) = shift;
    }
}

else
{
    return $self->($name);
}
```


Perl – OOP – Destructors

To free memory of the object

Called using DESTROY method

A destructor method is simply a member function (subroutine) named DESTROY, which will be called automatically in following cases –

- When the object reference's variable goes out of scope.

- When the object reference's variable is undef-ed.

- When the script terminates

- When the perl interpreter terminates

Perl – OOP – Destructors

```
package MyClass;
```

```
sub DESTROY
```

```
{
```

```
    print "MyClass::DESTROY called\n";
```

```
}
```