Perl is a programming language developed by Larry Wall, especially designed for text processing. It stands for **Practical Extraction and Report Language**.

Perl is Interpreted

Perl is an interpreted language, which means that your code can be run as is, without a compilation stage that creates a non portable executable program.

$perl -v

**Perl Official Website** − <https://www.perl.org/>

You can download Perl documentation from the following site.

**Perl Documentation Website** − [https://perldoc.perl.org](https://perldoc.perl.org/)

## Running Perl

The following are the different ways to start Perl.

### Interactive Interpreter

You can enter **perl** and start coding right away in the interactive interpreter by starting it from the command line. You can do this from Unix, DOS, or any other system, which provides you a command-line interpreter or shell window.

$perl -e <perl code> # Unix/Linux

or

C:>perl -e <perl code> # Windows/DOS

### Script from the Command-line

A Perl script is a text file, which keeps perl code in it and it can be executed at the command line by invoking the interpreter on your application, as in the following −

$perl script.pl # Unix/Linux

or

C:>perl script.pl # Windows/DOS

## First Perl Program

### Interactive Mode Programming

You can use Perl interpreter with **-e** option at command line, which lets you execute Perl statements from the command line. Let's try something at $ prompt as follows −

$perl -e 'print "Hello World\n"'

This execution will produce the following result −

Hello, world

### Script Mode Programming

Assuming you are already on $ prompt, let's open a text file hello.pl using vi or vim editor and put the following lines inside your file.

#!/usr/bin/perl

# This will print "Hello, World"

print "Hello, world\n";

Here **/usr/bin/perl** is actual the perl interpreter binary. Before you execute your script, be sure to change the mode of the script file and give execution priviledge, generally a setting of 0755 works perfectly and finally you execute the above script as follows −

$chmod 0755 hello.pl

$./hello.pl

This execution will produce the following result −

Hello, world

You can use parentheses for functions arguments or omit them according to your personal taste. They are only required occasionally to clarify the issues of precedence. Following two statements produce the same result.

print("Hello, world\n");

print "Hello, world\n";

## Comments in Perl

# This is a comment in perl

# This is a single line comment

print "Hello, world\n";

#$a=10;

#$b=20;

=begin comment

Test

test

=cut

## Single and Double Quotes in Perl

#!/usr/bin/perl

print "Hello, world\n";

print 'Hello, world\n';

Number

#!/usr/bin/perl

$a = 10;

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

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

## "Here" Documents

#!/usr/bin/perl

$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";

$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 - Data Types

|  |  |
| --- | --- |
| **Sr.No.** | **Types & Description** |
| 1 | **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. |
| 2 | **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 (@). |
| 3 | **Hashes**  Hashes are unordered sets of key/value pairs that you access using the keys as subscripts. They are preceded by a percent sign (%). |

## Numeric Literals

|  |  |
| --- | --- |
| **Type** | **Value** |
| Integer | 1234 |
| Negative integer | -100 |
| Floating point | 2000.99 |
| Scientific notation | 16.12E14 |
| Hexadecimal | 0xffff |
| Octal | 0577 |

## String Literals

|  |  |
| --- | --- |
| **Escape 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 |
| \0nn | Creates Octal formatted numbers |
| \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 |

## Creating Variables

## Scalar Variables

$age = 25; # An integer assignment

$name = "John Paul"; # A string

$salary = 1445.50; # A floating point

## Array Variables

#!/usr/bin/perl

@ages = (25, 30, 40);

@names = ("John Paul", "Lisa", "Kumar");

print "\$ages[0] = $ages[0]\n"; # ages[0] = 25

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

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

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

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

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

## Hash Variables

#!/usr/bin/perl

$ : scaler

@ : array

% : hash

%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";

## Variable Context

Perl treats same variable differently based on Context, i.e., situation where a variable is being used. Let's check the following example −

#!/usr/bin/perl

@names = ('John Paul', 'Lisa', 'Kumar');

@copy = @names;

$size = @names;

print "Given names are : @copy\n";

print "Number of names are : $size\n";

## Scalar Operations

#!/usr/bin/perl

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

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

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

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

print "str = $str\n";

print "num = $num\n";

print "mul = $mul\n";

print "mix = $mix\n";

## Multiline Strings

#!/usr/bin/perl

$string = 'This is

a multiline

string';

print "$string\n";

## Special Literals

#!/usr/bin/perl

Logcal ….

print "File name ". \_\_FILE\_\_ . "\n";

print "Line Number " . \_\_LINE\_\_ ."\n"; #5

cod….

print "Line Number " . \_\_LINE\_\_ ."\n"; #2

print "Package " . \_\_PACKAGE\_\_ ."\n";

# they can not be interpolated

print "\_\_FILE\_\_ \_\_LINE\_\_ \_\_PACKAGE\_\_\n";

## Array Creation

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

@array = qw/This is an array/;

## Accessing Array Elements

#!/usr/bin/perl

@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";

## Sequential Number 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

## Array Size

@array = (1,2,3);

print "Size: ",scalar @array,"\n";

#!/usr/bin/perl

@array = (1,2,3);

$array[50] = 4;

$size = @array;

$max\_index = $#array;

print "Size: $size\n";

print "Max Index: $max\_index\n";

## Adding and Removing Elements in Array

#!/usr/bin/perl

# 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";

# 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";

## Slicing Array Elements

#!/usr/bin/perl

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

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

print "@weekdays\n";

## Replacing Array Elements

#!/usr/bin/perl

@nums = (1..20);

print "Before - @nums\n";

splice(@nums, 5, 5, 21..25);

print "After - @nums\n";

## Transform Strings to Arrays

#!/usr/bin/perl

# 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

## Sorting Arrays

#!/usr/bin/perl

# define an array

@foods = qw(pizza steak chicken burgers);

print "Before: @foods\n";

# sort this array

@foods = sort(@foods);

print "After: @foods\n";

## The $[ Special Variable

We have a special variable, which is written as **$[**. This special variable is a scalar containing the first index of all arrays. Because Perl arrays have zero-based indexing, $[ will almost always be 0. But if you set $[ to 1 then all your arrays will use on-based indexing. It is recommended not to use any other indexing other than zero. However, let's take one example to show the usage of $[ variable –

#!/usr/bin/perl

# 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";

## Merging Arrays

#!/usr/bin/perl

@odd = (1,3,5);

@even = (2, 4, 6);

@numbers = (@odd, @even);

print "numbers = @numbers\n";

## Selecting Elements from Lists

#!/usr/bin/perl

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

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

# Perl - Hashes

## Creating Hashes

Hashes are created in one of the two following ways. In the first method, you assign a value to a named key on a one-by-one basis −

$data{'John Paul'} = 45;

$data{'Lisa'} = 30;

$data{'Kumar'} = 40;

#!/usr/bin/perl

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

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

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

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

## Extracting Keys and Values

#!/usr/bin/perl

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

@names = keys %data;

print "$names[0]\n";

print "$names[1]\n";

print "$names[2]\n";

## Checking for Existence

#!/usr/bin/perl

%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";

}

## Getting Hash Size

#!/usr/bin/perl

%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";

## Add and Remove Elements in Hashes

#!/usr/bin/perl

%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";