PERL - SCALARS

A scalar is a single unit of data. That data might be an integer number, floating point, a character, a string, a paragraph, or an entire web page.

Here is a simple example of using scalar variables −

```
#!/usr/bin/perl

$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
```

Numeric Scalars

A scalar is most often either a number or a string. Following example demonstrates the usage of various types of numeric scalars −

```
#!/usr/bin/perl

$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
```

String Scalars

Following example demonstrates the usage of various types of string scalars. Notice the difference between single quoted strings and double quoted strings −

```
#!/usr/bin/perl

$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!
```

Scalar Operations

You will see a detail of various operators available in Perl in a separate chapter, but here we are going to list down few numeric and string 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 "mix = $mix\n";
```

This will produce the following result –

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

Multiline Strings

If you want to introduce multiline strings into your programs, you can use the standard single quotes as below –

```
#!/usr/bin/perl

$string = 'This is
a multiline
string';

print "$string\n";
```

This will produce the following result –

```
This is
a multiline
string
```

You can use "here" document syntax as well to store or print multilines as below –

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

```
print <<EOF;
This is
a multiline
string
EOF
```

This will also produce the same result −

```
This is
a multiline
string
```

V-Strings

A literal of the form v1.20.300.4000 is parsed as a string composed of characters with the specified ordinals. This form is known as v-strings.

A v-string provides an alternative and more readable way to construct strings, rather than use the somewhat less readable interpolation form "\x{1}\x{14}\x{12c}\x{fa0}".

They are any literal that begins with a v and is followed by one or more dot-separated elements. For example −

```
#!/usr/bin/perl

$smile  = v9786;
$foo    = v102.111.111;
$martin = v77.97.114.116.105.110;

print "smile = $smile\n";
print "foo = $foo\n";
print "martin = $martin\n";
```

This will also produce the same result −

```
smile = ☺
foo = foo
martin = Martin
Wide character in print at /tmp/135911788320439.pl line 7.
```

Special Literals

So far you must have a feeling about string scalars and its concatenation and interpolation opration. So let me tell you about three special literals __FILE__, __LINE__, and __PACKAGE__ represent the current filename, line number, and package name at that point in your program.

They may be used only as separate tokens and will not be interpolated into strings. Check the below example −

```
#!/usr/bin/perl

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__
```