# PERL - VARIABLES

 Copyright © tutorialspoint.com

Variables are the reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or strings in these variables.

We have learnt that Perl has the following three basic data types −

- Scalars
- Arrays
- Hashes

Accordingly, we are going to use three types of variables in Perl. A **scalar** variable will precede by a dollar sign $ and it can store either a number, a string, or a reference. An **array** variable will precede by sign @ and it will store ordered lists of scalars. Finaly, the **Hash** variable will precede by sign % and will be used to store sets of key/value pairs.

Perl maintains every variable type in a separate namespace. So you can, without fear of conflict, use the same name for a scalar variable, an array, or a hash. This means that $foo and @foo are two different variables.

## Creating Variables

Perl variables do not have to be explicitly declared to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign = is used to assign values to variables.

> *Keep a note that this is mandatory to declare a variable before we use it if we use **use strict** statement in our program.*

The operand to the left of the = operator is the name of the variable, and the operand to the right of the = operator is the value stored in the variable. For example −

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

Here 25, "John Paul" and 1445.50 are the values assigned to *$age*, *$name* and *$salary* variables, respectively. Shortly we will see how we can assign values to arrays and hashes.

## Scalar Variables

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. Simply saying it could be anything, but only a single thing.

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
```

## Array Variables

An array is a variable that stores an ordered list of scalar values. Array variables are preceded by an "at" @ sign. To refer to a single element of an array, you will use the dollar sign $ with the variable name followed by the index of the element in square brackets.

Here is a simple example of using array variables −

```
#!/usr/bin/perl

@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";
```

Here we used escape sign (\) before the $ sign just to print it. Other Perl will understand it as a variable and will print its value. When exected, 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
```

## Hash Variables

A hash is a set of **key/value** pairs. Hash variables are preceded by a percent sign. To refer to a single element of a hash, you will use the hash variable name followed by the "key" associated with the value in curly brackets.

Here is a simple example of using hash variables −

```
#!/usr/bin/perl

%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
```

## 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";
```

This will produce the following result −

```
Given names are : John Paul Lisa Kumar
Number of names are : 3
```

Here @names is an array, which has been used in two different contexts. First we copied it into anyother array, i.e., list, so it returned all the elements assuming that context is list context. Next we used the same array and tried to store this array in a scalar, so in this case it returned just the number of elements in this array assuming that context is scalar context. Following table lists down the various contexts −

| S.N. | Context and Description |
|---|---|
| 1 | **Scalar −**<br><br>Assignment to a scalar variable evaluates the right-hand side in a scalar context. |
| 2 | **List −**<br><br>Assignment to an array or a hash evaluates the right-hand side in a list context. |
| 3 | **Boolean −**<br><br>Boolean context is simply any place where an expression is being evaluated to see whether it's true or false. |
| 4 | **Void −**<br><br>This context not only doesn't care what the return value is, it doesn't even want a return value. |
| 5 | **Interpolative −**<br><br>This context only happens inside quotes, or things that work like quotes. |

Loading [MathJax]/jax/output/HTML-CSS/jax.js