Array

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

In Perl, List and Array terms are often used as if they're interchangeable. But the list is the data, and the array is the variable.

Array Creation

Array variables are prefixed with the @ sign and are populated using either parentheses or the qw operator. For example −

```
@array = (1, 2, 'Hello');
@array = qw/This is an array/;
```

The second line uses the qw// operator, which returns a list of strings, separating the delimited string by white space. In this example, this leads to a four-element array; the first element is 'this' and last (fourth) is 'array'. This means that you can use different lines as follows −

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

```
...
Sunday/;
```

You can also populate an array by assigning each value individually as follows −

```
$array[0] = 'Monday';
...
$array[6] = 'Sunday';
```

Accessing Array Elements

When accessing individual elements from an array, you must prefix the variable with a dollar sign ($) and then append the element index within the square brackets after the name of the variable. For example −

```
#!/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";
```

This will produce the following result −

```
Mon
Tue
Wed
Sun
Sun
Mon
```

Array indices start from zero, so to access the first element you need to give 0 as indices. You can also give a negative index, in which case you select the element from the end, rather than the beginning, of the array. This means the following −

```
print $days[-1]; # outputs Sun
print $days[-7]; # outputs Mon
```

Sequential Number Arrays

Perl offers a shortcut for sequential numbers and letters. Rather than typing out each element when counting to 100 for example, we can do something like as follows −

```
#!/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
```

Here double dot (..) is called **range operator**. This will produce the following result −

```
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
```

Array Size

The size of an array can be determined using the scalar context on the array - the returned value will be the number of elements in the array −

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

The value returned will always be the physical size of the array, not the number of valid elements. You can demonstrate this, and the difference between scalar @array and $#array, using this fragment is as follows −

```
#!/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";
```

This will produce the following result −

```
Size: 51
Max Index: 50
```

There are only four elements in the array that contains information, but the array is 51 elements long, with a highest index of 50.

Adding and Removing Elements in Array

Perl provides a number of useful functions to add and remove elements in an array. You may have a question what is a function? So far you have used **print** function to print various values. Similarly there are various other functions or sometime called sub-routines, which can be used for various other functionalities.

Sr.No.	Types & Description

1	**push @ARRAY, LIST** Pushes the values of the list onto the end of the array.
2	**pop @ARRAY** Pops off and returns the last value of the array.
3	**shift @ARRAY** Shifts the first value of the array off and returns it, shortening the array by 1 and m everything down.
4	**unshift @ARRAY, LIST** Prepends list to the front of the array, and returns the number of elements in the new 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";
```

This will produce the following result −

1. @coins = Quarter Dime Nickel
2. @coins = Quarter Dime Nickel Penny
3. @coins = Dollar Quarter Dime Nickel Penny
4. @coins = Dollar Quarter Dime Nickel
5. @coins = Quarter Dime Nickel

Slicing Array Elements

You can also extract a "slice" from an array - that is, you can select more than one item from an array in order to produce another array.

```
#!/usr/bin/perl

@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
```

The specification for a slice must have a list of valid indices, either positive or negative, each separated by a comma. For speed, you can also use the **..** range operator −

```
#!/usr/bin/perl

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

@weekdays = @days[3..5];

print "@weekdays\n";
```

This will produce the following result −

```
Thu Fri Sat
```

Replacing Array Elements

Now we are going to introduce one more function called **splice()**, which has the following syntax −

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

This function will remove the elements of @ARRAY designated by OFFSET and LENGTH, and replaces them with LIST, if specified. Finally, it returns the elements removed from the array. Following is the example −

```
#!/usr/bin/perl

@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

Here, the actual replacement begins with the 6th number after that five elements are then replaced from 6 to 10 with the numbers 21, 22, 23, 24 and 25.

Transform Strings to Arrays

Let's look into one more function called **split()**, which has the following syntax −

split [PATTERN [, EXPR [, LIMIT]]]

This function splits a string into an array of strings, and returns it. If LIMIT is specified, splits into at most that number of fields. If PATTERN is omitted, splits on whitespace. Following is the example −

```
#!/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
```

This will produce the following result −

Roses
Michael

Transform Arrays to Strings

We can use the **join()** function to rejoin the array elements and form one long scalar string. This function has the following syntax −

join EXPR, LIST

This function joins the separate strings of LIST into a single string with fields separated by the value of EXPR, and returns the string. Following is the example −

```
#!/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);
```

```
$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
```

Sorting Arrays

The **sort()** function sorts each element of an array according to the ASCII Numeric standards. This function has the following syntax −

```
sort [ SUBROUTINE ] LIST
```

This function sorts the LIST and returns the sorted array value. If SUBROUTINE is specified then specified logic inside the SUBTROUTINE is applied while sorting the elements.

```
#!/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";
```

This will produce the following result −

```
Before: pizza steak chicken burgers
After: burgers chicken pizza steak
```

Please note that sorting is performed based on ASCII Numeric value of the words. So the best option is to first transform every element of the array into lowercase letters and then perform the sort function.

Merging Arrays

Because an array is just a comma-separated sequence of values, you can combine them together as shown below −

```
#!/usr/bin/perl

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

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

This will produce the following result −

numbers = 1 3 4 5 6

The embedded arrays just become a part of the main array as shown below −

```
#!/usr/bin/perl

@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

Selecting Elements from Lists

The list notation is identical to that for arrays. You can extract an element from an array by appending square brackets to the list and giving one or more indices −

```
#!/usr/bin/perl

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

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

This will produce the following result −

value of var = 1

Similarly, we can extract slices, although without the requirement for a leading @ character −

```
#!/usr/bin/perl

@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 Array with Loops

Perl array elements can be accessed within a loop. Diferent types of loops can be used.

We will show array accessing with following loops:

- foreach loop
- for loop
- while loop
- until loop

Perl Array with foreach Loop

In foreach loop, the control variable is set over the elements of an array. Here, we have specified $i as the control variable and print it.

```
@num = qw(10 20 30 40 50);
foreach $i (@num) {
  print "$i\n";
}
```

Output:

```
40
50
```

Perl Array with for Loop

A control variable will be passed in for loop as the index of the given array.

```
@num = qw(10 20 30 40 50);
for($i = 0; $i < 5; $i++){
  print "@num[$i]\n";
```

```
}
```

Output:

```
10
20
30
40
50
```

Perl Array with while Loop

The while loop executes as long as the condition is true.

```
$i = 5;
while ($i > 0) {
  print "$i\n";
  $i--;
}
```

Output:

```
5
4
3
2
1
```

Perl Array with until Loop

The until loop works like while loop, but they are opposite of each other. A while loop runs as long as a condition is true whereas an until loop runs as long as condition is false. Once the condition is false until loop terminates.

The until loop can be written on the right hand side of the equation as an expression modifier.

```
@your_name = "John";
print "@your_name\n" until $i++ > 4;
```

Output:

```
John
John
John
John
John
```

In the above program, once $i is greater than 4 according to the condition, loop iteration stops.

Perl Multidimensional Array

Perl multidimensional arrays are arrays with more than one dimension. The multi dimensional array is represented in the form of rows and columns, also called Matrix.

They can not hold arrays or hashes, they can only hold scalar values. They can contain references to another arrays or haashes.

Perl Multidimensional Array Matrix Example

Here, we are printing a 3 dimensional matrix by combining three different arrays **arr1**, **arr2** and **arr3**. These three arrays are merged to make a matrix array **final**.

Two for loops are used with two control variables **$i** and **$j**.

```
## Declaring arrays
my @arr1 = qw(0 10 0);
my @arr2 = qw(0 0 20);
my@arr3 = qw(30 0 0);
## Merging all the single dimensional arrays
my @final = (\@arr1, \@arr2, \@arr3);
print "Print Using Array Index\n";
for(my $i = 0; $i <= $#final; $i++){
   # $#final gives highest index from the array
   for(my $j = 0; $j <= $#final ; $j++){
```

```
    print "$final[$i][$j] ";
  }
  print "\n";
}
```

Output:

```
Print Using Array Index
0 10 0
0 0 20
30 0 0
```

Perl Multidimensional Array Initialization and Declaration Example

In this example we are initializing and declaring a three dimensional Perl array .

```
@array = (
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
      );
     for($i = 0; $i < 3; $i++) {
    for($j = 0; $j < 3; $j++) {
       print "$array[$i][$j] ";
  }
  print "\n";
}
```

Output:

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

Perl String Escaping Characters

All the special characters or symbols like @, #, $, & /, \, etc does not print in a normal way. They need a preceding escaping character backward slash (\) to get printed.

Perl Displaying E-mail Address

All the e-mail addresses contain (@) sign. As stated earlier, symbols will not be printed normally inside a string. They need extra attention. Use backward slash (\) before @ sign to print e-mail addresses.

```
1. use strict;
2. use warnings;
3. my $site  = "javatpoint\@gmail.com";
4. print "$site\n";
```

Output:

```
javatpoint@gmail.com
```

Perl Escaping Escape Character

If we want to print (\) sign inside a string, use backward slash (\) preceding \ sign.

```
1. use strict;
2. use warnings;
3. print "The \\n is a new line chracter in programming languages.\n";
```

Output:

```
Everyone has to follow this rule whether its a boy\girl
```

Perl Escaping Double quotes

If you want to print double quotes inside a string use backslash (\) at both the quotes.

```
1. use strict;
2. use warnings;
3. my $x = 'tutorials';
4. print "Our site \"javaTpoint\" provides all type of \"$x\"\n";
```

Output:

Perl Modules and namespaces

A module is a container which holds a group of variables and subroutines which can be used in a program. Every module has a public interface, a set of functions and variables.

To use a module into your program, **require** or **use** statement can be used, although their semantics are slightly different.

The 'require' statement loads module at runtime to avoid redundant loading of module. The 'use' statement is like require with two added properties, compile time loading and automatic importing.

Namespace is a container of a distinct set of identifiers (variables, functions). A namespace would be like **name::variable**.

Every piece of Perl code is in a namespace.

In the following code,

```
1. use strict;
2. use warnings;
3. my $x = "Hello";
4. $main::x = "Bye";
5. print "$main::x\n";  # Bye
6. print "$x\n";       # Hello
```

Here are two different variables defined as **x**. the **$main::x** is a package variable and **$x** is a lexical variable. Mostly we use lexical variable declared with my keyword and use namespace to separate functions.

In the above code, if we won't use **use strict**, we'll get a warning message as

```
1. Name    "main::x" used only once: possible typo at line..
```

The **main** is the namespace of the current script and of current variable. We have not written anything and yet we are already in the 'main' namespace.

By adding 'use strict', now we got the following error,

```
1. Global symbol "$x" requires explicit package name
```

In this error, we got a new word 'package'. It indicates that we forgot to use 'my' keyword before declaring variable but actually it indicates that we should provide name of the package the variable resides in.

Perl Hashes

The hashes is the most essential and influential part of the perl language. A hash is a group of key-value pairs. The keys are unique strings and values are scalar values.

Hashes are declared using my keyword. The variable name starts with a (%) sign.

Hashes are like arrays but there are two differences between them. First arrays are ordered but hashes are unordered. Second, hash elements are accessed using its value while array elements are accessed using its index value.

No repeating keys are allowed in hashes which makes the key values unique inside a hash. Every key has its single value.

Perl Hash Accessing

To access single element of hash, ($) sign is used before the variable name. And then key element is written inside {} braces.

```
1. my %capitals = (
2.     "India"  => "New Delhi",
3.     "South Korea" => "Seoul",
4.     "USA"  => "Washington, D.C.",
5.     "Australia"  => "Canberra"
6. );
```

```
print"$capitals{'India'}\n";
print"$capitals{'South Korea'}\n";
print"$capitals{'USA'}\n";
print"$capitals{'Australia'}\n";
```

Output:

```
New Delhi
Seoul
Washington, D.C.
Canberra
```

Perl Hash Indexing

Hashes are indexed using $key and $value variables. All the hash values will be printed using a while loop. As the while loop runs, values of each of these variables will be printed.

```
my %capitals = (
    "India"  => "New Delhi",
    "South Korea" => "Seoul",
    "USA"  => "Washington, D.C.",
    "Australia"  => "Canberra"
);
# LOOP THROUGH IT
while (($key, $value) = each(%capitals)){
     print $key.", ".$value."\n";
}
```

Output:

```
Australia, Canberra
India, New Delhi
USA, Washington, D.C.
South Korea, Seoul
```

Perl sorting Hash by key

You can sort a hash using either its key element or value element. Perl provides a sort() function for this. In this example, we'll sort the hash by its key elements.

```
my %capitals = (
    "India"  => "New Delhi",
    "South Korea" => "Seoul",
    "USA"  => "Washington, D.C.",
    "Australia"  => "Canberra"
);
# Foreach loop
foreach $key (sort keys %capitals) {
     print "$key: $capitals{$key}\n";
}
```

Output:

```
Australia: Canberra
India: New Delhi
South Korea: Seoul
USA: Washington: D.C.
```

Look at the output, all the key elements are sorted alphabetically.

Difference between hash and hash reference in perl

Hash

A hash is a basic data type in Perl. It uses keys to access its contents.

Hash reference

A hash ref is an abbreviation to a reference to a hash. References are scalars, that is simple values. It is a scalar value that contains essentially, a pointer to the actual hash itself.

```
my %hash = (
   toy    => 'aeroplane',
   colour => 'blue',
);
print "I have an ", $hash{toy}, " which is coloured ", $hash{colour}, "\n";
```

```
my $hashref = \%hash;
print "I have an ", $hashref->{toy}, " which is coloured ", $hashref->{colour}, "\n";
```