Perl Hash

Hashes, or hash tables, that are called **associative arrays**, **hashmaps**, or **dictionaries** in other languages are an integral and important part of Perl. On this page we try to answer some common questions about hashes.

Perl Hash table tutorial

A hash in Perl always starts with a percentage sign: %. When accessing an element of a hash we replace the % by a dollar sign $ and put curly braces {} after the name. Inside the curly braces we put the key.

A hash is an unordered set of key-value pairs where the keys are unique.

A key can be any string including numbers that are automatically converted to strings. A value can be any scalar value: number, string, or a reference.

The key is a string, but when it is a "simple string" you can leave out the quote characters when used on the left hand side of the fat-arrow, or in the curly braces.

```
1.  use strict;
2.  use warnings;
3.  use 5.010;
4.
5.  my %person = (
6.     fname => 'Foo',
7.     lname => 'Bar',
8.  );
9.  say $person{'fname'};  # Foo
10. say $person{fname};    # Foo
11. my $key = 'fname';
12. say $person{$key};     # Foo
```

Perl Hash of arrays

Each value in the following hash is an array, or more specifically it is a reference to an array.

```
1.  use strict;
2.  use warnings;
3.  use 5.010;
4.  use Data::Dumper qw(Dumper);
5.
```

```
6.  my %grades;
7.  $grades{'Foo Bar'}[0] = 23;
8.  $grades{'Foo Bar'}[1] = 42;
9.  $grades{'Foo Bar'}[2] = 73;
10. $grades{'Peti Bar'}[0] = 10;
11. $grades{'Peti Bar'}[1] = 15;
12. print Dumper \%grades;
13.
14. $grades{'Zorg'} = [10, 20, 30, 40];
15.
16. print Dumper \%grades;
```

In the first 5 lines we access the elements of the internal arrays as if we had a two-dimensional data structure. In the last assignment we assign an array reference [10, 20, 30, 40] to Zorg.

Data::Dumper can show the data structure in a reasonably readable way:

```
$VAR1 = {
          'Foo Bar' => [
                         23,
                         42,
                         73
                       ],
          'Peti Bar' => [
                          10,
                          15
                        ]
        };
$VAR1 = {
          'Foo Bar' => [
                         23,
                         42,
                         73
                       ],
          'Peti Bar' => [
```

```
                    10,
                    15
                  ],
        'Zorg' => [
                    10,
                    20,
                    30,
                    40
                  ]
      };
```

Perl Hash of arrays of arrays

Like in the preceding example, each value in the following hash is a reference to an array and each value in the array is a reference to another array.

Here is an example of a list of invoices for each customer:

```
1.  use strict;
2.  use warnings;
3.  use Data::Printer;
4.
5.  my $invoices = {
6.    customer_1 => [
7.        [ 1, 'Article_1', 300.00 ],
8.        [ 2, 'Article_2', 500.00 ],
9.    ],
10.   customer_2 => [
11.       [ 1, 'Article_2', 999.00 ],
12.       [ 2, 'Article_5', 399.99 ],
13.   ],
14. };
15.
16. # Add another customer
17. push @{ $invoices->{customer_3} }, [ 1, 'Article_9', 899.00 ];
18. push @{ $invoices->{customer_3} }, [ 2, 'Article_10', 799.00 ];
19.
20. p $invoices;
```

This time we use Data::Printer to show the data structure:

```
\ {
    customer_1   [
        [0] [
            [0] 1,
            [1] "Article_1",
            [2] 300
        ],
        [1] [
            [0] 2,
            [1] "Article_2",
            [2] 500
        ]
    ],
    customer_2   [
        [0] [
            [0] 1,
            [1] "Article_2",
            [2] 999
        ],
        [1] [
            [0] 2,
            [1] "Article_5",
            [2] 399.99
        ]
    ],
    customer_3   [
        [0] [
            [0] 1,
            [1] "Article_9",
            [2] 899
        ],
        [1] [
            [0] 2,
            [1] "Article_10",
            [2] 799
        ]
    ]
}
```

Note the different format used by this module.

Perl Hash reference

```
1.  use strict;
2.  use warnings;
3.  use 5.010;
4.  use Data::Dumper qw(Dumper);
5.
6.  my %phones = (
7.     Foo => '1-234',
8.     Bar => '1-456',
9.  );
10. my $hr = \%phones;
11.
12. say $phones{Foo};  # 1-234
13. say $hr->{Foo};    # 1-234
14.
15. print Dumper $hr;
16.
17. foreach my $name (keys %$hr) {
18.    say "$name $hr->{$name}";
19. }
20.
21. my $other_ref = {
22.    Qux => '1-567',
23.    Moo => '1-890',
24. };
25.
26. say $other_ref->{Qux};   # 1-567
27. print Dumper $other_ref;
```

```
1-234
1-234
$VAR1 = {
          'Foo' => '1-234',
          'Bar' => '1-456'
        };
Foo 1-234
Bar 1-456
1-567
```

```
$VAR1 = {
          'Qux' => '1-567',
          'Moo' => '1-890'
        };
```

Perl Hash key

Hashes are key-value pairs. Let's say we have a hash called %phone_number_of. If you know a specific key, which is just a string, and it is found in the variable $name, then you can get the value of this key in the above hash by writing $phone_number_of{$name}.

If you don't know what keys are in the hash you can fetch a list of keys using @names = keys %phone_number_of.

Perl Hash exists

Given an expression that specifies an element of a hash, returns true if the specified element in the hash has ever been initialized, even if the corresponding value is undefined.

A hash element can be true only if it's defined and defined only if it exists, but the reverse doesn't necessarily hold true.

```
1.  use strict;
2.  use warnings;
3.
4.  my %months = (
5.      0 => 'January',
6.      1 => 'February',
7.      2 => 'March',
8.      3 => 'April',
9.      4 => 'May',
10.     5 => 'June',
11.     6 => 'July',
12.     7 => 'August',
13.     8 => 'September',
14.     9 => 'October',
15.     10 => 'November',
16.     11 => 'December'
17. );
18.
```

```
19. #Interpolation will not happen for hashes i.e %months will not
    be interpolated
20. if (exists $months{1}) {
21.     print "$months{1} exists in the hash %months\n";
22. }
23.
24. my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
    localtime(time);
25. print "The current month is $months{$mon}" if exists
    $months{$mon};
```

Perl Hash size

In this hash, keys contain multiple words (i.e 2 words), so you need to enclose it in quotes. If the key contains only a single word, then quotes are optional. In fact, it is recommended to omit quotes for keys.

```
1.  use strict;
2.  use warnings;
3.
4.  #Program to find the size of a hash
5.
6.  my %india = (
7.      'National Bird'   => 'Peacock',
8.      'National Animal' => 'Tiger',
9.      'National Flower' => 'Lotus',
10.     'National Fruit'  => 'Mango',
11.     'National Tree'   => 'Banyan',
12.     'National Game'   => 'Hockey'
13. );
14.
15. #The keys function in scalar context returns the number of keys
    in the hash.
16. my $size = keys %india;
17.
18. print "The size of the hash is $size\n";
```

Perl hash number of elements

See above at **Perl Hash size**

Perl Hash map

Perl Hash slice

A slice is always a list, so the hash slice notation uses an at sign to indicate that. The curly braces mean that you're indexing into a hash; the at sign means that you're getting a whole list of elements, not just a single one (which is what the dollar sign would mean).

```
1.  use strict;
2.  use warnings;
3.
4.  use 5.010;
5.
6.  my %employee = (
7.      jack  => 980144,
8.      peter => 128756,
9.      john  => 903610
10. );
11.
12. #Assign a hash slice to @id1 array
13. my @id1 = ($employee{"jack"}, $employee{"peter"},
    $employee{"john"});
14.
15. #Print all employee ids from array @id1
16. say join ',', @id1;
17.
18. #Assign a hash slice to @id2 array
19. my @id2 = @employee{ qw/jack peter john/ };
20.
21. #Print all employee ids from array @id2
22. say join ',', @id2;
23.
24. my %employee2 = (
25.     #Name, Employee Id, Department, Location
26.     jack  => [980144,'Marketing','London'],
27.     peter => [128756,'Research', 'Detroit'],
28.     john  => [903610, 'Development', 'Sydney']
29. );
30.
31. #Retrieve the location of all employees
32. my @location = ($employee2{"jack"}->[2], $employee2{"peter"}-
    >[2], $employee2{"john"}->[2]);
33.
34. #Print all employee's location
35. say join ',', @location;
```

Hash slices are a very useful feature of Perl that remove the need for some loops. A hash slice is a way of referring to one or more elements of the hash in one statement, to get a list of values, or to assign a list of values.

To get a single element from a hash %hash, with key $key, you can write $value = $hash{ $key }

To get a list of elements from the same hash, referred to by the keys in @keys, you can write @values = @hash{ @keys }

```
1.  use strict;
2.  use warnings;
3.
4.  #Program to demonstrate hash slice
5.
6.  my %day_names = (
7.     'sun'  => 'Sunday',
8.     'mon'  => 'Monday',
9.     'tue'  => 'Tuesday',
10.    'wed'  => 'Wednesday',
11.    'thu'  => 'Thursday',
12.    'fri'  => 'Friday',
13.    'sat'  => 'Saturday',
14. );
15.
16. #Get a list of the full names of week days (ie not weekends)
17. my @weekdays = @day_names{ qw(mon tue wed thu fri) };
18.
19. print "The store is open from 9AM to 5PM on " . join(", ",
    @weekdays) . "\n";
20.
21. #Get a list of the full names of weekend days
22. my @weekends = @day_names{ 'sat', 'sun' };
23.
24. print "The store closes at 12 noon on " . join(" and ",
    @weekends) . "\n";
25.
26. #Lets say we want to change the hash now to make the values
    lower case and plural
27. #So that 'Sunday' becomes 'sundays'
28. #We can assign to a hash slice to achieve this
29.
30. #Get the keys and the values from the hash - these will have
    the same respective order
31. my @keys = keys %day_names;
```

```
32. my @values = values %day_names;
33.
34. #Now assign to a slice of the hash %day_names
35. #In this case the slice @keys identifies every key of
   %day_names
36. @day_names{ @keys } = map lc($_) . 's', @values;
37.
38. print "In the winter the store may open late " .
   $day_names{sun} . "\n";
```

Size of an array in a hash

Getting the size of an array within a hash is a matter of de-referencing it @{ $data{$key} } and putting that in scalar context either explicitly: scalar @{ $data{$key} }, or one of the many implicit ways: $count = @{ $data{$key} }, if (@{ $data{$key} } < 10) {

```
1.  use strict;
2.  use warnings;
3.  use 5.010;
4.
5.  my %data = (
6.    Snowwhite => [ 'Doc', 'Grumpy', 'Happy', 'Sleepy', 'Bashful',
   'Sneezy', 'Dopey' ],
7.    LOTR      => [ 'Frodo', 'Sam Gamgee', 'Pippin', 'Merry',
   'Aragorn', 'Boromir', 'Legolas', 'Gimli', 'Gandalf'],
8.  );
9.  say scalar @{ $data{Snowwhite} };    # 7
10. my $dwarfs = @{ $data{Snowwhite} };
11. my $fellowship = @{ $data{LOTR} };
12. say $dwarfs;                         # 7
13. say $fellowship;                     # 9
```

Number of elements of an array in a hash

This is the same as the **size of an array in a hash**.