Unique values in an array reference in Perl

unique uniq distinct filter grep array

In an earlier article we saw several solutions to filter out duplicate values from an array, but what if our data is in a reference to an array?

What if we have our data in an array reference like the `$words` in this example:

```
1. my @words_array = qw(foo bar baz foo zorg baz);
   my $words = \@words_array;
```

The solution involves quite a few brackets and braces, but it can still be expressed in one expression:

```
1. my $unique = uniq($data);
   sub uniq { [ keys { map { $_ => 1 } @{$_[0]} } ] };
```

The `uniq` function we created expects a reference to an array and returns a reference to an array.

Before we try to explain the code, let's see if it really works!

Let's create a script with the data and solution we saw earlier:

```
 1. use strict;
    use warnings;
    use Data::Dumper;

 5. my @words_array = qw(foo bar baz foo zorg baz);
    my $data = \@words_array;

    my $unique = uniq($data);
    sub uniq { [ keys { map { $_ => 1 } @{$_[0]} } ] };
10.
11. print Dumper $unique;
```

The output will be the following (though the order of the words can be different on every run):

```
$VAR1 = [
          'bar',
          'foo',
```

```
          'zorg',
          'baz'
        ];
```

B::Deparse

We have already seen the use of B::Deparse. Let's see if it can help us:

I have redureced the `uniq.pl` script to include only the function we try to understand:

```
1. sub uniq { [ keys { map { $_ => 1 } @{$_[0]} } ] };
```

And ran the following:

```
$ perl -MO=Deparse uniq.pl

keys on reference is experimental at uniq.pl line 1.
sub uniq {
    [keys {map({$_, 1;} @{$_[0];})}];
}
```

as I can see, it added parentheses around the parameters of `map` and added `;` in a few places, but it did not help a lot.

It also gave a warning: `keys on reference is experimental at uniq.pl line 1.`

Let's try B::Deparse with some more parameters:

Adding the `-p` parameter gave us even more parantheses:

```
$ perl -MO=Deparse,-p uniq.pl

keys on reference is experimental at uniq.pl line 1.
sub uniq {
    [keys({map({($_, 1);} @{$_[0];})})];
}
```

I am not sure this helped.

Understanding one expression at a time

Let's go back to the original expression and let' try to take it apart to understand step-by-step:

```
1. sub uniq { [ keys { map { $_ => 1 } @{$_[0]} } ] };
```

We could move the expression in the subroution to a separate row and move the closing braces to a third row:

```
1. sub uniq {
       [ keys { map { $_ => 1 } @{$_[0]} } ]
   };
```

A subroutine receives its paramaters in the `@_` array. The first element of that array is `$_[0]`.

(Just as the first element of `@names` would be `$names[0]`.)

We assume this is a reference to an array and want to de-reference it so we can go over the elements one-by-one. We do that by putting a `@` in front of the reference.

Explanation: If we had an array reference in `$ar` then we would write `@$ar` to dereference it. Because we have the array reference in `$_[0]` we could just put the `@` at the beginning: `@$_[0]`, but then it would not be clear: Did we want to de-reference `$_[0]`; or did we want to de-reference `$_` getting `@$_`, and then wanted to create an array-slice with the first element only? In order to make this crystal clear we put curly braces around the expressions that belong together. That's why we write `@{$_[0]}`.

Now that we have the list of values in an array we can use `map` on the array. That's how we get the expressions `map { $_ => 1 } @{$_[0]}`. This map returns a list of values. Each odd value would be one of the elements of the original array, and each even value would be the number 1.

The curly braces around the `map` convert these values into a hash reference in which the keys will be the values from the original array.

Starting from perl 5.14 the `keys` function can work on hash references as well, not only on hashes, but B::Deparse warns about this. That's the warning `keys on reference is experimental` we saw earlier.

Finally the `keys` function returns a list. We wrap that in square brackets in order to convert the list into an array reference. This is what the `uniq` function returns.