undef on Perl arrays and hashes

undef delete defined

When employing `undef` on scalar variable, you can write it in two ways, and they have the same effect.

When you do it on an array or a hash, it will be different. Let's try to clear up the confusion.

undef of scalar variables

Check out these two code snippets:

The first one has `$x = undef;`:

```
use strict;
use warnings;

my $x = 42;
$x = undef;

print defined $x ? 'DEFINED' : 'NOT';
```

and the second one uses `undef $x;`

```
use strict;
use warnings;

my $x = 42;
undef $x;

print defined $x ? 'DEFINED' : 'NOT';
```

Both will print "NOT". `$x = undef` and `undef $x` are exactly the same. They are also the same as `$x = undef()` and `undef($x)`, just in case you like parentheses.

undef on array elements

Try this script which has `$names[1] = undef;` in it:

```
use strict;
use warnings;

use Data::Dumper qw(Dumper);

my @names = qw(Foo Bar Baz);
$names[1] = undef;

print Dumper \@names;
```

It will print the following:

```
$VAR1 = [
          'Foo',
          undef,
          'Baz'
        ];
```

Replacing `$names[2] = undef;` by `undef $names[2];` yields the same result as those two calls are the same.

delete on arrays

`delete $names[2];` is deprecated and likely to be removed in a future version of Perl. To delete the 3rd element of an array (index 2) use `splice(@names, 2, 1)`. Then go and read more about splice.

undef on arrays

We will try this code now, calling `undef @names;`

```
use strict;
use warnings;

use Data::Dumper qw(Dumper);

my @names = qw(Foo Bar Baz);
undef @names;

print Dumper \@names;
```

```
$VAR1 = [];
```

The array became empty.

We can replace `undef @names;` by `@names = ();` and we get he same result. An empty array.

On the other hand, if we use `@names = undef;` that will leave the array with a single element which is undef.

```
$VAR1 = [
          undef
        ];
```

This is **NOT what you want**!

undef on hash elements

The script uses `$h{Foo} = undef;` to set the value of a hash key to be `undef`.

```
1. use strict;
   use warnings;

   use Data::Dumper qw(Dumper);
5.
   my %h = (Foo => 123, Bar => 456);
   $h{Foo} = undef;

   print Dumper \%h;
```

Will set the value of Foo in the %h hash to be `undef`:

```
$VAR1 = {
          'Bar' => 456,
          'Foo' => undef
        };
```

`undef $h{Foo};` would do exactly the same.

delete a hash element

Writing `delete $h{Foo};` instead of the call to `undef` will remove both the key and the value from the hash:

```
$VAR1 = {
          'Bar' => 456
        };
```

Putting `delete` on the other side does not make sense at all: `$h{Foo} delete;` is a syntax error.

undef on a whole hash

See this `undef %h;` in the following code:

```
1. use strict;
   use warnings;

   use Data::Dumper qw(Dumper);
5.
   my %h = (Foo => 123, Bar => 456);
   undef %h;

   print Dumper \%h;
```

```
$VAR1 = {};
```

Writing `%h = ()` instead of `undef %hl` will also make the hash empty just as above.

On the other hand writing `%h = undef;` is incorrect. It will generate the following output:

```
Odd number of elements in hash assignment at files/eg.pl line 7.
Use of uninitialized value in list assignment at files/eg.pl line 7.
$VAR1 = {
          '' => undef
        };
```

It looks a bit odd. What happened here is that the `undef` we typed in was converted to an empty string generating the Use of uninitialized value in list assignment at ... warning. This became the key in the hash.

Then there was no corresponding value. This generated the **Odd number of elements in hash assignment** warning, and an `undef` was assigned to be the value of the empty-string key.

In any case, this is **NOT what you want**!

As a conclusion let me try to answer to straight forward question:

How do you reset an array and a hash in Perl?

```
1. @a = ();
   %h = ();
```

How do you reset a complete hash or a hash key/value pair?

Reset complete hash:

```
1. %h = ();
```

Remove a key/value pair:

```
1. delete $h{Foo};
```

Remove only the value of a key/value pair:

```
1. $h{Foo} = undef;
```