ref - What kind of reference is this variable?

ref SCALAR ARRAY HASH CODE REF GLOB LVALUE FORMAT IO VSTRING Regexp

The `ref()` function will return the type of the reference it got as a parameter. If no parameter was supplied, it will return the reference type of $_, the default variable of Perl.

According to the documentation, the possible return values of the `ref()` function are:

```
SCALAR
ARRAY
HASH
CODE
REF
GLOB
LVALUE
FORMAT
IO
VSTRING
Regexp
```

Let's see when do we get such values:

Simple scalars

If we pass a simple scalar variable to the `ref()` function containingundef, a string, or a number, the `ref()` function will return the empty string:

```perl
use strict;
use warnings;
use 5.010;

my $nothing;
my $string = 'abc';
my $number = 42;

say 'nothing:    ', ref $nothing;   #
say 'string:      ', ref $string;     #
say 'number:      ', ref $number;     #
say 'nothing:     ', defined ref $nothing;    # 1
say 'string:      ', defined ref $string;     # 1
say 'number:      ', defined ref $number;     # 1
```

Reference to SCALAR

If we take the reference to either of the simple scalars, event the undef, the `ref()` function will return the string `SCALAR`.

```
use strict;
use warnings;
use 5.010;

my $nothing;
my $string = 'abc';
my $number = 42;

my $nothingref = \$nothing;
my $stringref  = \$string;
my $numberref  = \$number;

say 'nothingref: ', ref $nothingref; # SCALAR
say 'stringref:  ', ref $stringref;  # SCALAR
say 'numberref:  ', ref $numberref;  # SCALAR
```

Reference to ARRAY and HASH

If we pass an array or a hash to the `ref()` it will return an empty string, but if we pass a reference to an array, or a reference to a hash, it will return `ARRAY`, or `HASH` respectively.

```
use strict;
use warnings;
use 5.010;

my @arr = (2, 3);
my %h = (
    answer => 42,
);

my $arrayref  = \@arr;
my $hashref   = \%h;
```

```
say 'array:      ', ref @arr;       #
say 'hash:       ', ref %h;         #
say 'arrayref:   ', ref $arrayref;  # ARRAY
say 'hashref:    ', ref $hashref;   # HASH
```

Reference to CODE

Passing a reference to a subroutine to the `ref()` function will result in the string `CODE`.

```
use strict;
use warnings;
use 5.010;

sub answer {
     return 42;
}
my $subref    = \&answer;

say 'subref:     ', ref $subref;    # CODE
```

A reference to a reference: REF

If we have a reference to a reference, and we pass that to the `ref()` function, it will return the string `REF`.

```
use strict;
use warnings;
use 5.010;

my $str = 'abc';
my $strref = \$str;
my $refref    = \$strref;
say 'strref:     ', ref $strref;    # SCALAR
say 'refref:     ', ref $refref;    # REF

say 'refrefref:  ', ref \$refref;   # REF
```

Even if we have a reference to a reference to a reference..... that will be still `REF`.

Reference to a Regex

The `qr` operator returns a pre-compiled regular expression, or if you ask the `ref()` function, then `qr` returns a reference to a `Regexp`.

```
use strict;
use warnings;
use 5.010;

my $regex = qr/\d/;
my $regexref = \$regex;
say 'regex:       ', ref $regex;     # Regexp

say 'regexref:    ', ref $regexref;  # REF
```

Of course if we take a reference to the `Regex` reference we are back to the `REF` as above.

Reference to GLOB

A file-handle created by the `open` function is a `GLOB`.

```
use strict;
use warnings;
use 5.010;

open my $fh, '<', $0 or die;
say 'filehandle: ', ref $fh;         # GLOB
```

Reference to a FORMAT

I think the format function of Perl fell out of favor by most of the Perl developers and you can rarely see it in the wild. I could not even figure out how to take a reference to it in a simple way, but let me leave the example here as it is. You probably don't need to worry about it.

```
use strict;
use warnings;
use 5.010;

format fmt =
   Test: @<<<<<<<< @||||| @>>>>>
.
say 'format:      ', ref *fmt{FORMAT};  # FORMAT
```

Reference to VSTRING

Version string staring with the letter `v`, are another rare sighting, even though they are more used than formats:

```
use strict;
use warnings;
use 5.010;

my $vs = v1.1.1;
my $vsref = \$vs;
say 'version string ref: ', ref $vsref;  # VSTRING
```

Reference to LVALUE

Lvalue functions are functions that can appear on the left hand side of an assignment. For example if you would like to change the content of a string you can use the 4-parameter version of substr, the 4th parameter being the replacement string, or you can assign that string to the 3-parameter version of substr.

Let's see what happens if we take a reference of a regular, 4-parameter substr call:

```
use strict;
use warnings;
use 5.010;

my $text = 'The black cat climbed the green tree';
my $nolv = \ substr $text, 14, 7, 'jumped from';
say 'not lvalue:  ', ref $nolv;  # SCALAR
say $nolv;    # SCALAR(0x7f8d190032b8)
say $$nolv;   # climbed
say $text;    # The black cat jumped from the green tree

$$nolv = 'abc';
say $text;    # The black cat jumped from the green tree
```

The value assigned to the `$nolv` variable is a regular reference to a scalar containing the value returned by the `substr` function. The word 'climbed' in this case.

On the other hand, if we take a reference to a 3-parameter substr call (or 2-parameter for that matter), then the returned value that gets assigned to `$lv` below, is a reference to an `LVALUE`. If we de-reference it `say $$lv;`, we can see the original value (the string 'climbed') in it.

If we assign to that dereference `$$lv = 'jumped from';` that will change the content of `$$lv`, but that will also replace the selected part in `$text`, the original string.

We can repeated this assignment: `$$lv = 'abc';` that will change the original string again.

```
use strict;
use warnings;
use 5.010;
```

```
my $text = 'The black cat climbed the green tree';
my $lv = \ substr $text, 14, 7;
say 'lvalue:      ', ref $lv;     # LVALUE
say $lv;                          # LVALUE(0x7f8fbc0032b8)
say $$lv;                         # climbed
say $text;                        # The black cat climbed the green tree

$$lv = 'jumped from';
say $lv;                          # LVALUE(0x7f8fbc0032b8)
say $$lv;                         # jumped from
say $text;                        # The black cat jumped from the green tree

$$lv = 'abc';
say $$lv;                         # abc
say $text;                        # The black cat abc the green tree
```

Blessed references

As explained elsewhere, in the classic object oriented system of Perl the `bless` function is used to connect a hash reference to a namespace. (Actually it is the same in Moo and Moose, but there it is mostly hidden from our eyes.)

Anyway, if we call the `ref()` on a blessed reference, it will return the namespace it has been blessed into:

```
use strict;
use warnings;
use 5.010;

my $r = {};
say ref $r;               # HASH
bless $r, 'Some::Name';
say ref $r;               # Some::Name
```

The same even if the underlying reference is not a hash reference:

```
use strict;
use warnings;
use 5.010;

my $r = [];
say ref $r;                # ARRAY
bless $r, 'Class::Name';
```

```
    say ref $r;                 # Class::Name
```

More

The documentation of perlref has a lot more details about the `ref` function and about references in general.