wantarray - returning list or scalar based on context

wantarray

You are already familiar with the LIST and SCALAR context of Perl and while you might still remember how confusing it was when you first encountered it, but you feel that function you are currently writing would greatly benefit from that flexibility.

In other words, you'd like to create a function that will return different values depending on the context it was used in.

For example you'd like to create a function that given N, a whole number will compute the first N values of the Fibonacci function. In scalar context it will return the value of `fibo(N)`. In list context it will return the whole series of N numbers.

VOID context

Before implementing it, we have to mention one more thing, that has not been stressed earlier. Perl actually has 3 major contexts.

LIST, SCALAR and VOID context. This last one is created when the value of an expression is not used, when a function is called but no one checks the return value. The most common statements we usually use in VOID context are `print`, `say`, and `chomp`. It is quite rare in Perl code that someone checks if a call to `print` was successful, and checking the return value of `chomp` usually happens to people who don't yet understand how chomp works.

So the three contexts are created like this:

```
my @series = fibo($n);   # LIST
my $value  = fibo($n);   # SCALAR
fibo($n);                # VOID
```

wantarray

When the developer of the `fibo` function implements the function, she cannot know in which context it will be executed, but she can use the `wantarray` built-in inside the definition of the `fibo` function to check the current context of each call.

`wantarray` itself has 3 different return values:

- it can return undef
- it can return some false value, other than undef.
- it can return true value

These return values map to the 3 cases of how the function in which we use `wantarray` was called:

wantarray will return

- undef if the function was called in VOID context
- false, other than undef if the function was called in SCALAR context
- true if the function was called in LIST context.

Here is the skeleton of the code with 3 different invocations:

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

sub fibo {
    if (wantarray) {
        say 'LIST';
    } elsif (defined wantarray) {
        say  'SCALAR';
    } else {
        say  'VOID';
    }
}

my @numbers = fibo();  # LIST
my $value   = fibo();  # SCALAR
fibo();                # VOID
```

And now let's implement the Fibonacci:

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

sub fibo {
    my ($n) = @_;

    die 'There is no point in calling fibo() in VOID context'
        if not defined wantarray;

    my @fibo = (1, 1);
    push @fibo, $fibo[-1] + $fibo[-2] for 3 .. $n;

    return wantarray ? @fibo : $fibo[-1];
}

my @numbers = fibo(4);
say "@numbers";
my $value   = fibo(5);
say $value;
fibo(100);
```

It is written slightly differently from the skeleton. The first thing we check is if `wantarray` returns undef. If it does, meaning the user was not interested in the return value, we can assume it was a mistake. At this point we can just call `return` without making any noise. We could call `warn` and then return, or, as in the above example, we can call `die` and throw an exception.

The next two lines are the lines where we actually calculate the first N numbers of the Fibonacci series.

Then comes the call to `return` either the whole array, or just the last element using the ternary operator.

The result looks like this:

```
$ perl fibonacci.pl
1 1 2 3
5
There is no point in calling fibo() in VOID context at fibonacci.pl line 9.
```

Conclusion and comments

While I think the context sensitivity in Perl is very interesting, it can take time till a new Perl programmer comprehends it. Adding such behavior to user-defined function can make the API of a module better, but it can also make it harder to understand. **Use it with caution!**

The name `wantarray` is unfortunately misleading. For many new Perl programmers it takes some time to understand what is the difference between ARRAYS and LISTS, the fact that the function called `wantarray` actually checks if the enclosing function was called in LIST context just adds to this confusion. On the other hand, beginner Perl programmers probably should not use the `wantarray` function.

Finally, Perl actually has more fine-grained contexts, for example string and numeric context. In another article we'll see how we can create a function that will aware of those contexts as well.