NAME

List::Util - A selection of general-utility list subroutines

SYNOPSIS

```
use List::Util qw(first max maxstr min minstr reduce shuffle sum);
```

DESCRIPTION

`List::Util` contains a selection of subroutines that people have expressed would be nice to have in the perl core, but the usage would not really be high enough to warrant the use of a keyword, and the size so small such that being individual extensions would be wasteful.

By default `List::Util` does not export any subroutines. The subroutines defined are

first BLOCK LIST

Similar to `grep` in that it evaluates BLOCK setting `$_` to each element of LIST in turn. `first` returns the first element where the result from BLOCK is a true value. If BLOCK never returns true or LIST was empty then `undef` is returned.

```
    $foo = first { defined($_) } @list    # first defined value in
@list
    $foo = first { $_ > $value } @list    # first value in @list
which
                                          # is greater than $value
```

This function could be implemented using `reduce` like this

```
    $foo = reduce { defined($a) ? $a : wanted($b) ? $b : undef }
undef, @list
```

for example wanted() could be defined() which would return the first defined value in @list

max LIST

Returns the entry in the list with the highest numerical value. If the list is empty then `undef` is returned.

```
    $foo = max 1..10                # 10
    $foo = max 3,9,12               # 12
    $foo = max @bar, @baz           # whatever
```

This function could be implemented using `reduce` like this

```
    $foo = reduce { $a > $b ? $a : $b } 1..10
```

maxstr LIST

Similar to `max`, but treats all the entries in the list as strings and returns the highest string as defined by the `gt` operator. If the list is empty then `undef` is returned.

```
    $foo = maxstr 'A'..'Z'          # 'Z'
    $foo = maxstr "hello","world"   # "world"
    $foo = maxstr @bar, @baz        # whatever
```

This function could be implemented using `reduce` like this

```
    $foo = reduce { $a gt $b ? $a : $b } 'A'..'Z'
```

min LIST

Similar to `max` but returns the entry in the list with the lowest numerical value. If the list is empty then `undef` is returned.

```
    $foo = min 1..10                # 1
    $foo = min 3,9,12               # 3
    $foo = min @bar, @baz           # whatever
```

This function could be implemented using `reduce` like this

```
    $foo = reduce { $a < $b ? $a : $b } 1..10
```

minstr LIST

Similar to `min`, but treats all the entries in the list as strings and returns the lowest string as defined by the `lt` operator. If the list is empty then `undef` is returned.

```
    $foo = minstr 'A'..'Z'          # 'A'
    $foo = minstr "hello","world"   # "hello"
    $foo = minstr @bar, @baz        # whatever
```

This function could be implemented using `reduce` like this

```
    $foo = reduce { $a lt $b ? $a : $b } 'A'..'Z'
```

reduce BLOCK LIST

Reduces LIST by calling BLOCK, in a scalar context, multiple times, setting `$a` and `$b` each time. The first call will be with `$a` and `$b` set to the first two elements of the list, subsequent calls will be done by setting `$a` to the result of the previous call and `$b` to the next element in the list.

Returns the result of the last call to BLOCK. If LIST is empty then `undef` is returned. If LIST only contains one element then that element is returned and BLOCK is not executed.

```
    $foo = reduce { $a < $b ? $a : $b } 1..10       # min
    $foo = reduce { $a lt $b ? $a : $b } 'aa'..'zz' # minstr
    $foo = reduce { $a + $b } 1 .. 10               # sum
    $foo = reduce { $a . $b } @bar                  # concat
```

If your algorithm requires that `reduce` produce an identity value, then make sure that you always pass that identity value as the first argument to prevent `undef` being returned

```
  $foo = reduce { $a + $b } 0, @values;             # sum with 0
identity value
```

shuffle LIST

Returns the elements of LIST in a random order

```
    @cards = shuffle 0..51      # 0..51 in a random order
```

sum LIST

Returns the sum of all the elements in LIST. If LIST is empty then `undef` is returned.

```
    $foo = sum 1..10                # 55
    $foo = sum 3,9,12               # 24
    $foo = sum @bar, @baz           # whatever
```

This function could be implemented using `reduce` like this

```
    $foo = reduce { $a + $b } 1..10
```

If your algorithm requires that `sum` produce an identity of 0, then make sure that you always pass `0` as the first argument to prevent `undef` being returned

```
  $foo = sum 0, @values;
```

KNOWN BUGS

With perl versions prior to 5.005 there are some cases where reduce will return an incorrect result. This will show up as test 7 of reduce.t failing.

SUGGESTED ADDITIONS

The following are additions that have been requested, but I have been reluctant to add due to them being very simple to implement in perl

```
# One argument is true

sub any { $_ && return 1 for @_; 0 }

# All arguments are true

sub all { $_ || return 0 for @_; 1 }

# All arguments are false

sub none { $_ && return 0 for @_; 1 }

# One argument is false

sub notall { $_ || return 1 for @_; 0 }

# How many elements are true

sub true { scalar grep { $_ } @_ }

# How many elements are false

sub false { scalar grep { !$_ } @_ }
```

SEE ALSO

Scalar::Util, *List::MoreUtils*

COPYRIGHT