# Scalars.

... their identifiers are in separate namespaces. There can be an array with the same name as a scalar.

It is truncated.

Strings and arrays can be any size. Contrast with Java arrays which are of limited size.

If you exceed at the end by assignment, the array is automatically expended. Any "empty slots" that need to be created are filled with `undef`.
If you exceed at the beginning, or at the end by lookup, `undef` is returned.

```
$#ident
```

As in Python, they index the set starting from the end.

The range operator (`..`) placed between two numbers. The numbers are truncated.

```
qw( no quotes necessary whitespace delimited )
```

Any punctuation can be used in place of the parens.

- You can't put comments inside.
- If you want to use the delimiter as an element of the list you must backslash escape.

```
($id1, ..., $idN) = (val1, ..., valN)
```
The RHS need not be a list *literal*.

If there are too many values they will be ignored. If there are too many variables the extra variables will get their default values.

The resulting list will be flattened, containing the elements of any "nested list" along with any scalar values in the literal.

Copies the whole array.

It's slow.

`pop` which mutates the array and returns the last element (or `undef`)
`push` which takes an array and a scalar or two arrays, mutating the first

`shift` is like popping from the beginning.
`unshift` is like pushing to the beginning (*cons*).

They are printed with the elements with spaces interposed.

The same as it was before the loop (*i.e.*, quite possibly undefined).

You can leave off the control variable. The default `$_` will be used.

They use `$_` instead.

It returns a new array with the argument reversed. The original argument is not mutated.

Returns a sorted array, using ascii order. Does not mutate the argument.

List and scalar.

`undef` will result.

A qualifier.

Context, the location in the program text where the expression occurs.

Bare list variables evaluate to lists only in list context. They evaluate to the size of the list in scalar context.

An expression that would be more surprising if it were used in scalar context than if it was used in list context.

List context. That's a list of length one, not a scalar with extraneous parens.

List.

The scalar will be wrapped into a singleton list.

Use the `scalar` operator.

In scalar context, the next line of input. In list context, all remaining lines.

It chomps each element, in place.

```
chomp(@lines = <FILEHANDLE>)
```