shuffle

syntax: `shuffle(data)`
`shuffle(data,nsamps)`

`shuffle(urn,0)`
`shuffle(urn,-nsamps)`

purpose: Scrambles randomly the order of a data set, analogous to shuffling a deck of cards. This can be used for sampling *without* replacement.

`shuffle(data,n)` takes `n` samples from `data`, without replacement. If `n` is larger than the number of points in `data`, the sampling is done with replacement.

If data is a matrix, the sampling is done row-by-row, as in RESAMP.

You can also use `shuffle` to sample from a two column matrix that has integer multiplicities in the first column, and values in the second column. This format is described in SAMPLE. In order to signal to `shuffle` that the two column matrix is meant to be interpreted as multiplicities and values, and not just data, you should use `shuffle(matrix,0)` to take the same number of points as given by the sum of the multiplicities, or `shuffle(matrix, -n)` to sample `n` points from the matrix of multiplicities. (Or, you can use EXPAND.)

examples: Since SHUFFLE works randomly, you will likely get different results than these.

```
≫  x = [3 3 3 1 1 1];
≫  shuffle(x)
   ans:    3 1 1 3 1 3
≫  shuffle(x)
   ans:    1 3 3 1 1 3
```

When dealing with a matrix, the sampling is done row by row.

```
datamatrix = [1 1.3;
              2 2.4;
              3 3.5;
              4 4.6;]
```

```
≫  shuffle(datamatrix)
   ans:    4.0 4.6
           2.0 2.4
           1.0 1.3
           3.0 3.5
```

If your matrix has two columns, and reflects multiplicities and values, you need to signal this to `shuffle` by making the second argument 0 or negative. For instance, if we interpret `datamatrix` in this way, then it is equivalent to the vector `[1.3 2.4 2.4 3.5 3.5 3.5 4.6 4.6 4.6 4.6]`

```
≫  shuffle(datamatrix,0)
   ans:    2.4 3.5 2.4 4.6 3.5 1.3 4.6 3.5 4.6 4.6
≫  shuffle(datamatrix,-3)
   ans:    4.6 1.3 3.5
≫  shuffle(expand(datamatrix),4)
   ans:    1.3 2.3 4.6 3.5
```

see also: RESAMP, SAMPLE, EXPAND