Iterators Tutorial

`for` statement iterators were covered in the ForTutorial. These are some notes on writing your own custom iterators.

Algorithm

We can write our own *iterators* to be called by the `for` statement. The following Lua pseudo-code describes how `for` statement works with iterators:

```
do
  local iterator, state, var1 = explist
  local var2, ... , varn
  while true do
    var1, ..., varn = iterator(state, var1)
    if var1 == nil then break end
      -- for block code
  end
end
```

The state and current key value are passed into the iterator. The iterator returns the new value of the key and any other values, e.g. the value generated by the iterator. If `nil` is returned then the `for` loop is terminated.

Simple example

The following iterator will return a sequence of squared values. When we return no value (i.e. `n>=state`), Lua is returning `nil` which will terminate iteration. Notice the iterator is returning the *next* value in the sequence for a given iteration. We use the `state` to hold the number of iterations we wish to perform.

```
> function square(state,n) if n<state then n=n+1 return n,n*n end end
```

Here is the `for` statement calling the iterator:

```
> for i,n in square,5,0 do print(i,n) end
1       1
2       4
3       9
4       16
```

```
5       25
```

We could wrap the above example up (like `pairs()`) and provide a `squares(nbvals)` iterator constructor function. E.g.,

```
> function squares(nbvals) return square,nbvals,0 end  -- iterator,state,initial value
```

Now we can call it like `pairs()`:

```
> for i,n in squares(5) do print(i,n) end
1       1
2       4
3       9
4       16
5       25
```

FindPage · RecentChanges · preferences
edit · history
Last edited April 24, 2003 6:55 pm PDT (diff)