Programming in Lua – Iterators

Fabio Mascarenhas

http://www.dcc.ufrj.br/~fabiom/lua

Generic for

- We have seen how to use the *generic* `for` loop (or the for-in loop) using the `ipairs` and `pairs` functions, but there is nothing special about those functions

- The Lua standard library defines other functions that work with the generic for:

```
-- for each line in "foo.txt" do...
for line in io.lines("foo.txt") do
  -- for each word in line do...
  for word in string.gmatch(line, "%w+") do
    print(word)
  end
  print("------------")
end
```

- All these functions have one thing in common: they return *iterators*

Iterators

- An *iterator* is a function that, each time it is called, produces one or more values that correspond to an item from some sequence
  - Each index and value of an array ipairs
  - Each key and value from a table pairs
  - Each line from a file io.lines
  - Each substring that matches a pattern string.gmatch
- When there are no more items the iterator returns `nil`

Generic `for` and iterators

- The generic for takes the iterator returned by the calls to `ipairs`, `pairs`, `io.lines`, and `string.gmatch`, and repeatedly calls it, assigning the values it returns to the control variables

```
> iter = function ()
>>          local x = math.random(4)
>>          if x == 4 then
>>            return nil
>>          else
>>            return x
>>          end
>>        end
> for n in iter do print(n) end
1
3
1
```

Closure iterators

- The simplest way to define an useful iterator is to use a *closure*:

```
function fromto(a, b)
  return function ()
           if a > b then
             return nil
           else
             a = a + 1
             return a - 1
           end
         end
end
```

STATE OF THE ITERATOR

- The closure that `fromto` returns is the iterator:

```
> for i in fromto(2, 5) do print(i) end
2
3
4
5
```

Stateless iterators

- If we inspect the return values of ipairs, we see that it does not return just the iterator function:

```
> print(ipairs{ 1, 4, 5 })
function: 0000000068B94970      table: 00000000003EBE90 0
> print(ipairs{ 3, 9 })
function: 0000000068B94970      table: 00000000003EC020 0
```

- Moreover, it is returning the *same* iterator function for both both uses, so it cannot be using a closure closing over its parameter

- What ipairs returns is a *stateless* iterator, its *external state*, and its *seed*

- Each iteration, the generic for calls the iterator passing both the state and the seed, and then uses the value of the first control variable as a new seed

Stateless fromto

- We can define fromto using an stateless iterator if we use *b* as the state and the predecessor of *a* as the seed:

```
function fromto(a, b)
  return function (state, seed)
           if seed >= state then
             return nil
           else
             return seed + 1
           end
         end, b, a-1
end
```

not a true closure!

- Notice that the iterator function does not close over any variables, as both `state` and `seed` are parameters

```
> print(fromto(2, 5))
function: 0000000000420840      5       1
> print(fromto(4, 7))
function: 0000000000420840      7       3
```

Seedless iterator

- A variant of the stateless iterator uses a mutable value (a table, a file…) as the state, so it does not need a seed; the state keeps track where in the iteration we are

- This is analogous to the Java concept of iterators, as the call to `next` in a Java iterator has an implicit parameter (`this`)

```
function fromto(a, b)
  return function (state)
           if state[1] > state[2] then      → not a closure!
             return nil
           else
             state[1] = state[1] + 1
             return state[1] - 1
           end
         end, { a, b }
end
```

```
> print(fromto(2, 5))
function: 000000000042B0B0      table: 0000000000426160
> print(fromto(4, 7))
function: 000000000042B0B0      table: 0000000000426340
```

Quiz

- The function values returns an iterator, what does it produce? How can we turn it from a closure to a stateless iterator?

```
function values(t)   - iterator maker
  local i = 0
  return function ()
           i = i + 1
           return t[i]
         end
end
```

see the code
for class 4
for answer!