Programming in Lua – Functions

Fabio Mascarenhas

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

Functions

- A function abstracts and parameterizes a sequence of statements and expressions
- A function call can be a statement or an expression, depending on whether we are interested in which values a function returns (if any) or just its side effects

```
> print(8*9, 9/8)          → statement
72      1.125
> x = math.sin(3) + math.cos(10)     EXPRESSIONS
> print(x, os.date())
-0.69795152101659       06/23/13 14:49:03
```

Functions are values

- Functions are values, and have no exclusive namespace, so you have to be careful to not shadow built-in functions

```
> print(print)
function: 0000000068B94B40
> do
>> local print = 10
>> print(print)
>> end
stdin:3: attempt to call local 'print' (a
number value)
stack traceback:
        stdin:3: in main chunk
        [C]: in ?
```

Defining functions

- You can define new **local** functions very easily:

```lua
local function max(a, b)
  return (a > b) and a or b
end
```

- Local functions are local to the chunk, just like local variables
- The body of a function is a chunk, and the parameters are local variables in this chunk
- A `return` statement returns from a function

“Global” functions

- You can also write function definitions without the `local` keyword; in this case, the function gets assigned to the specified variable:

```
function max(a, b)
  return (a > b) and a or b
end
```

- If there is no local variable named `max` in scope then the function gets assigned to the global variable `max`

- To see the difference, the following is equivalent to the definition in the previous slide:

```
local max
function max(a, b)
  return (a > b) and a or b
end
```

Anonymous functions

- A third way of defining functions is anonymously, with an expression instead of a statement:

```lua
local max = function (a, b)
  return (a > b) and a or b
end
```

- All Lua functions are in fact anonymous, and “defining” a named function is actually assigning an anonymous function to a variable (global or local)

- Using function definition statements instead of plain assignment and anonymous functions is better programming style

Multiple results

- Functions can return multiple results; this is very useful with multiple assignment, or when calling functions that expect multiple parameters

```
> s, e = string.find("hello Lua users", "Lua")
> print(s, e)
7       9
> print(string.find("hello Lua users", "Lua"))
7       9
```

- Just list all return values to `return`:

```
function maxmin(a, b)
  if a < b then
    return b, a
  else
    return a, b
  end
end
```

Using multiple results

- Whenever a function call appears last in a list of expressions, Lua will append all of the results of the function to the list

```
> a, b = maxmin(2, 3)
> print(a, b)
3       2
> a, b, c = 1, maxmin(2, 3)
> print(a, b, c)
1       3       2
> print("maxmin(2, 3)", maxmin(2, 3))
maxmin(2, 3)    3       2
> function f(x) return maxmin(x, 0) end
> print(f(-4))
0       -4
```

- Otherwise, Lua uses just the first result and ignores the rest (if there are no results, Lua uses `nil`)

Using multiple results (2)

- Using just a single result from a function that returns multiple results:

```
> a, b = maxmin(2, 3), 5
> print(a, b)
3       5
> print(maxmin(2, 3), 5)
3       5
> print(maxmin(2, 3) + 2)
5
> a, b = (maxmin(2,3))
> print(a, b)
3       nil
> print((maxmin(2,3)))
3
```

Variadic functions

- The last parameter of a function can be the special token `...`
- All of the arguments passed to the function from the position of the `...` parameter forward will be produced by `...` inside the function

```
> function id(...) return ... end
> print(id(1, 2, 3))
1       2       3
```

```
function printf(fmt, ...)
  io.write(string.format(fmt, ...))
end
```

```
> printf("%s(%d, %d)\n", "maxmin", 2, 3)
maxmin(2, 3)
```

Quiz

- What is the output of the following program?

```lua
local function range(a, b, c)
  if a > b then
    return
  else
    return a, range(a + c, b, c)
  end
end
print(range(1, 9, 2))

-- The result is: 1       3       5       7       9
```