LabLua
programming language research

Programming in Lua – Control Flow

Fabio Mascarenhas

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

if-then-else

- An `if` statement executes the `then` chunk if the condition is *true* and the `else` chunk if it is *false*

```
if a < 0 then
  print("a is negative")
  a = -a
else
  print("a is positive")
end
```

- The `else` chunk is optional

- Remember that the condition does not need to be a boolean value, any value will do

elseif

- You can use `elseif` instead of writing nested `if` statements, thus avoiding a having to write `end` multiple times:

```
if op == "+" then
  r = a + b
elseif op == "-" then
  r = a - b
elseif op == "*" then
  r = a * b
elseif op == "/" then
  r = a / b
else
  error("invalid operation")
end
```

- The else part remains optional

while and repeat

- A `while` loop keeps executing its body chunk **while** the condition is *true*; Lua tests the condition *before* executing the body, so a `while` loop can run zero times

```
i = 1; sum = 0
while i <= 5 do
  sum = sum + (2 * i - 1)
  i = i + 1
end
print(sum)
```

- A `repeat` loop keeps executing its body chunk **until** the condition is *true*; Lua tests the condition *after* executing the body, so a `repeat` loop runs at least once

```
i = 1; sum = 0
repeat
  sum = sum + (2 * i - 1)
  i = i + 1
until i > 5
print(sum)
```

Numeric for

- A numeric `for` loop iterates a control variable from a *starting number* to an *ending number*, executing the body chunk

```
sum = 0
for i = 1, 5 do
  sum = sum + (2 * i - 1)
end
print(sum)
```

- The control variable is local to the body, so if you need its value after the loop it is better to use a `while` or `repeat` loop and explicitly manage the control variable

- The control variable cannot be assigned to, either, use a `break` statement if you want to terminate the loop early

Numeric for (2)

- If you pass a third number to the numeric for, it will add this number to the control variable after each iteration instead of 1

```
sum = 0
for i = 5, 1, -1 do
  sum = sum + (2 * i - 1)
end
print(sum)
```

- You can use expressions for the starting, ending, and step values, but they are evaluated only once before the loop starts

- A for loop can execute zero times, if the starting value is already greater than the ending value (or lesser than in case the step is negative)

Local variables

- A `local` statement declares a variable that is visible from the next statement to the end of the current chunk

```
local sum = 0              -- local to the program
for i = 1, 5 do
  local n = 2 * i - 1      -- local to the for body
  sum = sum + n
end
print(sum, n)
```

- New local variables *shadow* variables of the same name, whether global or local

- It is good style to use local variables whenever possible, and a common idiom to *cache* the value of a global variable in a local variable of the same name

do-end

- Entering the three statements of the previous slide in REPL does not do what we want, because each will be its own chunk

- But if we surround them in a do statement it will work

- You can use a do statement to introduce new scopes without changing control flow

```
sum = 0
do
  local i = 1
  while i <= 5 do
    sum = sum + (2 * i - 1)
    i = i + 1
  end
end
print(sum)
```

Multiple assignment

- Lua can assign to several different variables in a single step with *multiple assignment*

```
> a, b = 10, 2 * sum
> print(a, b)
10      50
```

- Lua first evaluates all expressions on the right side, then does the assignments, so you use multiple assignment to swap values

```
> a, b = b, a
> print(a, b)
50      10
```

Multiple assignment (2)

- If there are more variables than values to assign, `nil` gets assigned to the “extra” variables

```
> a, b, sum = 10, 2 * sum
> print(a, b, sum)
10      50      nil
```

- If there are more values than variables, the “extra” values are ignored

- Multiple assignments are very useful used in combination with functions that return multiple values

- A `local` statement can declare and initialize several local variables, and it works just like multiple assignment

Quiz

- What is the result of running the following program? Why?

```
local i = 5
while i do
  print(i)
  i = i - 1
end
```