CS 488 Tutorial

University of Waterloo
May 10 & 11, 2017

Lua

- Interpreted language

- Weak, dynamic typing

- Can be run interactively

- Can be extended

Example

```lua
-- This is an example Lua program
print("Hello, World!")

-- prints "1"
print(math.sin(math.pi)^2 + math.cos(math.pi)^2)

-- puts entered text into x
print("Enter some text:")
x = io.read()
print("You entered '" .. x .. "'.")
```

Comments

Single line comments start with --

```
print("Hello, World!") -- prints "Hello, World!"
```

Multi-line comments delimited by --[[and --]]

```
--[[

print("Goodbye!")

--]]
```

Common Data Types

nil: `nil`

boolean: `true`, `false`

number: `42`, `3.14`, `-0.3e+5`

string: `"Hello, World", 'Goodbye'`
```
[[
This is a multiline
string literal.
]]
```

table: `{}`, `{1,2,3}`, `{name = "Fred", age = 42}`

function: `function square(n) return n^2 end`

Operators

Arithmetic: binary: +, -, *, /, ^, unary: -

Relational: <, >, <=, >=, ==, ~=

   Take care with types: "0" == 0 is *false*, as is "2" < "15"

Logical: **and**, **or**, **not**

   and and **or** are short-circuiting

Concatenation: `..`

Access: `[]`, `.`

   Example:

```
a = {"x", "y", "z"}; print(a[2]) -- prints "y"
a[1] = 2
a["name"] = "Fred"
a.name = "Fred"
```

Assignment

```
a = 1
a, b = 1, 2
a, b = b, a        -- swap a and b
a[i], a[j], a[k] = a[k], a[i], a[j]


a, b, c = 0        -- 0, nil, nil
a, b, c = 0, 1     -- 0, 1, nil


a, b = 0, 1, 2     -- 2 ignored
```

Conditionals

```
if a > b then a = 0 end

if a > b then x = a else x = b end

if a > b then
  x = 1
  y = 2
end

if a > b then
  x = 1
elseif a < b
  x = -1
else
  x = 0
end
```

Loops

```
x = 1024
c = 0
while x > 1 do
 x = x / 2
 c = c + 1
end
print(c)

repeat
 line = io.read()
until line ~= ""
print(line)
```

Numeric For Loops

- Syntax:
  ```
  for var = start, stop [, step] do
      ...
    end
  ```
- `step` is assumed to be `1` if not specified

```
for i = 1, 10 do
  print(i)
end
```

```
for i = 99, 1, -1 do
  print(i .. " bottles of beer on the wall.")
  ...
end
```

Generic For Loops

- Syntax:

```
for var [, var ...] in iterator do
    ...
end
```

```
x = {5, 3, 9}
for i, v in ipairs(x) do
  print("x[" .. i .. "] = " .. v)
end
```

```
x = {name = "Fred", age = 42}
for k, v in pairs(x) do
  print("x." .. k .. " = " .. v)
end
```

Functions

```
function factorial(n)
  if n == 1 then
    return 1
  else
    return n * factorial(n-1)
  end
end
```

- Note: **return** may only be the last statement in a block

- *Workaround:* do return end

Functions

- Functions may return multiple values

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

a, b = sort(1, 2)
a, b = sort(2, 1)
```

Objects

```
-- returns a new material:
mat = gr.material(...)

-- returns a new sphere node:
node = gr.sphere('torso')

-- sets the material member:
node:set_material(mat)
```

Lua/C API

- C representation of lua commands have a common signature:

```
int command(lua_State *L);
```

- `L` maintains stack containing parameters

- Lua return values to be placed on the stack

- C function return value is the number of values returned

- API functions provided for manipulating stack

- Stack indexing:
  - Positive values start from bottom of the stack
  - Negative values start from the top
  - Zero is invalid

Lua/C API

```c
int gr_node_translate_cmd(lua_State* L)
{
   gr_node_ud* selfdata = (gr_node_ud*)
                        luaL_checkudata(L, 1, "gr.node");
   luaL_argcheck(L, selfdata != 0, 1, "Node expected");

   SceneNode *self = selfdata->node;
   double values[3];
   for (int i = 0; i < 3; i++) {
      values[i] = luaL_checknumber(L, i + 2);
   }

   self->translate(Vector3D(values[0], values[1], values[2]));
   return 0;
}
```

For call: red = gr.material({255, 0.0, 0.0}, {25, 25, 25}, 10)
We have C code:

```c
int gr_material_cmd(lua_State* L)
{
  gr_material_ud* data = (gr_material_ud*)
                          lua_newuserdata(L, sizeof(gr_material_ud));
  data->material = 0;

  luaL_checktype(L, 1, LUA_TTABLE);
  luaL_argcheck(L, luaL_len(L, 1) == 3, 1, "Three-tuple expected");
  luaL_checktype(L, 2, LUA_TTABLE);
  luaL_argcheck(L, luaL_len(L, 2) == 3, 2, "Three-tuple expected");
  luaL_checktype(L, 3, LUA_TNUMBER);

  double kd[3], ks[3];    // unpack the tables
  for (int i = 1; i <= 3; i++) {
    lua_rawgeti(L, 1, i);     kd[i - 1] = luaL_checknumber(L, -1);
    lua_rawgeti(L, 2, i);     ks[i - 1] = luaL_checknumber(L, -1);
    lua_pop(L, 2);
  }
  double shininess = luaL_checknumber(L, 3);
  data->material = new PhongMaterial(QVector3D(kd[0], kd[1], kd[2]),
                                     QVector3D(ks[0], ks[1], ks[2]),
                                     shininess);

  luaL_newmetatable(L, "gr.material");    lua_setmetatable(L, -2);
  return 1;
}
```

Lua/C API

```c
double kd[3], ks[3];
for (int i = 1; i <= 3; i++)
{
  lua_rawgeti(L, 1, i);
  kd[i - 1] = luaL_checknumber(L, -1);
  lua_rawgeti(L, 2, i);
  ks[i - 1] = luaL_checknumber(L, -1);
  lua_pop(L, 2);
}
```

Questions?

References:

Lua Home: http://www.lua.org/

Lua Users Wiki: http://lua-users.org/wiki/

Programming in Lua: http://www.lua.org/pil/

Changes in Lua 5.2

```
luaL_reg();  ->  luaL_Reg();

luaL_getn(L, #);  ->  luaL_len(L, #);

luaL_open()  ->  luaL_newstate();

luaL_openlib(L, 0, funcs, 0); -> luaL_setfuncs(L, funcs, 0);

luaL_openlib(L, "gr", funcs, 0);  ->

  luaL_setfuncs(L, funcs, 0); lua_setglobal(L, "gr");
```