Programming Fundamentals

Lecture 02 – Introduction to Lua

Edirlei Soares de Lima
<edirlei.lima@universidadeeuropeia.pt>

Lua Programming Language

- Lua is a powerful, efficient, lightweight, embeddable scripting language.
  - It supports procedural programming, object-oriented programming, functional programming, ...

- In video game development, Lua is one of the most popular scripting language for game programming.
  - Some games that use Lua: World of Warcraft, Civilization V, Far Cry, Angry Birds, Grim Fandango, Dota 2, ...

- Lua is designed, implemented, and maintained by a team of researchers at PUC-Rio in Brazil.

Why choose Lua?

- **Lua is a proven, robust language:**
  - Has been used in many industrial applications (e.g.: Adobe's Photoshop Lightroom) and games (e.g.: World of Warcraft and Angry Birds).
  - Is the leading scripting language in games and won the Front Line Award 2011 from the Game Developers Magazine.
  - Has a solid reference manual and several books.

- **Lua is fast:**
  - Lua has a deserved reputation for performance. Several benchmarks show Lua as the fastest language in the realm of interpreted scripting languages.

Why choose Lua?

- **Lua is portable:**
  - is distributed in a small package and builds out-of-the-box in all platforms that have a standard C compiler.
  - Lua runs on Unix, Windows, mobile devices (Android, iOS, BREW, Symbian, Windows Phone), on embedded microprocessors, etc.

- **Lua is embeddable:**
  - Lua can be easily embedded into other applications. Lua API allows strong and easy integration with code written in other languages.

- **Lua is free:**
  - Lua is a free open-source software that can be used for any purpose, including commercial purposes, at absolutely no cost.

Example of Code in Lua

```
local cels
local fahr

io.write("Temperature in Celsius: ")
cels = io.read()

fahr = 1.8 * cels + 32

io.write("Temperature in Fahrenheit: ", fahr, "\n")
```

Variables and Constants

- **Variables** and **constants** are the basic elements manipulated by a program.

- **Constant** is a fixed value that doesn't change during the execution of a program.

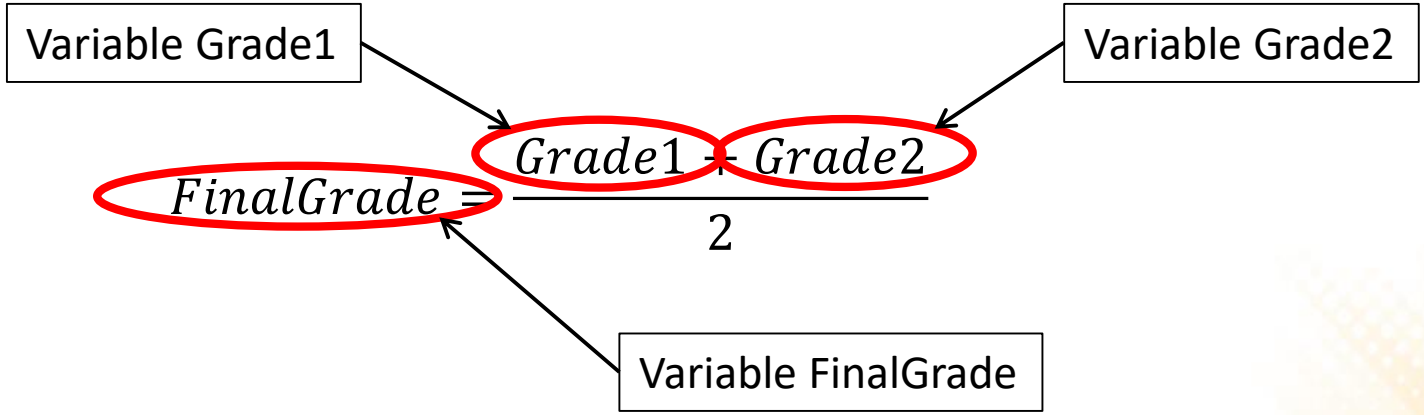
$$FinalGrade = \frac{Grade1 + Grade2}{2}$$

Contant value

Variables

- **Variable** is a space in the memory of the computer that is reserved to store a specific type of data.
  - Containers where we can store information (numbers, text, etc.)

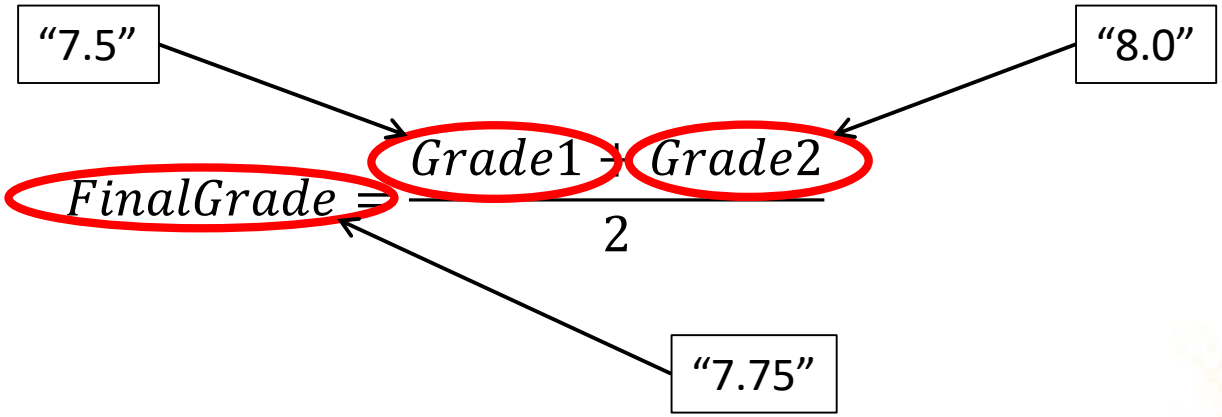
- Variables have **<u>names</u>** so they can be referenced in the code and have their values accessed or changed when necessary.



Variables

- The content of a variable can change during the execution of a program.

- Although different values can be assigned to the same variable, it can only store **one value at a time**.



Variables

- **Variables have:**
  - Name: used to refer to the variable in the code;
    - **Name restrictions:** is not allowed to start the name of variable with a number (0-9), some special characters are not allowed in the names (*, -, /, +, ...), and some reserved word can not be used as well (if, for, while, ...).
  - Type: defines the set of values that can be stored in the variable;
  - Value: the value stored;

- **Variables must be:**
  - Declared: What is the name and what is the type of the variable?
  - Initialized: What is the initial value of the variable?

Variables in Lua

- Lua is a **dynamically typed language**. This means that when a variable is declared, its type doesn't need to be specified.
  - There are no type definitions in Lua;
  - Each value carries its own type.

Type	Examples of Values
`number`	0, 1, 2.3, -2.3
`string`	"hi", "hello world", "test 123", ""
`boolean`	true, false
`function`	0x1234567
`table`	0x2345678
`thread`	0x3456789
`userdata`	0x4567890
`nil`	nil

Declaring Variables in Lua

- Local variables must be explicitly declared;
- More than one variable can be declared at a time;
- Variables can be used without being declared (global variables);

Examples:

```
local a      -- declares a local variable called a
local b      -- declares a local variable called b
local d, e   -- declares two local variables (a and b)
local d = 5  -- declares and initializes the variable d
f = 10       -- initializes a global variable f
```

Arithmetic Operators

- **Arithmetic Operators** are used to perform arithmetic operations with variables and constants.

Operation	Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder or modulus	%
Exponentiation	^

Examples:

assignment operator

total = price * quantity

final_grade = (grade1 + grade2)/2

result = 3 * (1 - 2) + 4 * 2

res = 4 % 2

res = b ^ 2

Input and Output Functions

- **Function "write" of the "io" module:** is used to write data to the output console.

```
io.write(constants/variables/expressions...)
```

```
io.write(33)

Output:
33
```

```
local myVar = 5
io.write("Value = ", 33, " Total = ", 33 + 40, " Var = ", myVar)

Output:
Valor = 33 Total = 73 Var = 5
```

Input and Output Functions

- Text output:

```
io.write("Programming Fundamentals\nwith Lua")

Output:
Programming Fundamentals
with Lua
```

Input and Output Functions

- **Function "read" of the "io" module:** is used to read data from the console (keyboard input).

```
io.read()
```

```
local n
n = io.read()
The value typed by the user is stored in the variable n
```

- **Important:** the input value is always a **string** (text). Sometimes you need to convert the value to a number type with the function `tonumber`:

```
local n
n = tonumber(io.read())
```

Example 1

- **Problem:** *read two numbers and show the sum of the numbers.*

```
local number1, number2, result

io.write("First number: ")
number1 = io.read()

io.write("Second number: ")
number2 = io.read()

result = number1 + number2

io.write("The sum is ", result)
```

In this case, we don't need to convert the values to numbers because arithmetic operators automatically convert them.

Lua Programming - Example

- **Comments:**

```
-- Program to convert temperatures from Celsius to Fahrenheit

local cels   -- variable to store the temperature in Celsius
local fahr   -- variable to store the temperature in Fahrenheit


io.write("Temperature in Celsius: ")
cels = io.read()  -- read the temperature in Celsius


fahr = 1.8 * cels + 32  -- convert from Celsius to Fahrenheit


-- Show the temperature in Fahrenheit
io.write("Temperature in Fahrenheit: ", fahr, "\n")
```

Exercises

1) Write a program that converts kilometers per hour to miles per hour (1 km/h is equal to 0.6213711922 mi/h).

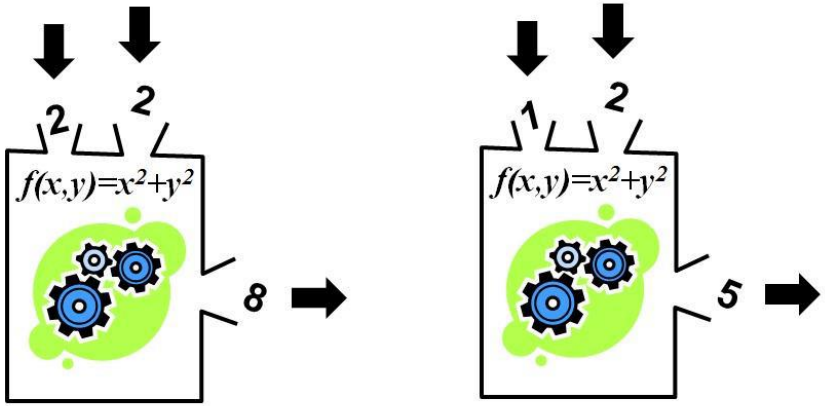
2) Write a program that calculates the perimeter of a rectangle ($P = 2(lenght + width)$).

3) Write a program that takes hours and minutes as input, and calculates the total number of minutes.

4) Write a program that takes minutes as input, and display the total number of hours and minutes.

Functions

- A function is a block of code with a name that can be executed at other points in the code.
  - It may have parameters
  - It may return a result



- **Functions are important to:**
  - Simplify and organize the code (modularization);
  - Avoid repetitions of code;
  - Extend the programming language;
  - Once declared, we can just use them (abstraction)

Functions in Lua

A Lua program cannot have two functions with the same name.

```
function function_name(parameter1, parameter2)
  local variables

  Lua instructions

  return
end
```

If a function doesn't have parameters, we just use: ()

Block of code

Functions in Lua – Example

```
function celsius_fahrenheit(tc)
  local f
  f = 1.8 * tc + 32
  return f
end

local cels, fahr
io.write("Temperatura in Celsius: ")
cels = io.read()
fahr = celsius_fahrenheit(cels)
io.write("Temperature in Fahrenheit: ", fahr)
```

We can use the function “celsius_fahrenheit” in any other program where this conversion may be needed.

Parameters and Return Values

- Example:

```
function celsius_fahrenheit(tc)
```

Only one input parameter.

- Example of function with two parameters:

```
function volume_cylinder(r, h)
  local v
  v = math.pi * (r ^ 2) * h
  return v
end
```

Two input parameters

Parameters and Return Values

```
function volume_cylinder(r, h)
  local v
  v = math.pi * (r ^ 2) * h
  return v
end

local radius, height, volume
io.write("Radius of the cylinder: ")
radius = io.read()
io.write("Height of the cylinder: ")
height = io.read()

volume = volume_cylinder(radius, height)

io.write("Volume of the cylinder: ", volume)
```

Scope of Variables

- The scope of a variable is the region of the program where the variable is valid/exist.

- A variable declared inside the block of code of a function with the keyword “local” is a **local variable**:
  - **The variable is only valid inside the block of code of the function where it was declared.**
  - When the execution of the function ends, the memory area reserved to store local variables is automatically released, so the program can no longer access those variables.

Scope of Variables

- **Local variable**:
  - A function can be executed multiple times.
    - For each execution, new memory areas for local variables are automatically reserved. When the function ends, the memory is automatically released.
  - Local variables declared inside the block of code of a function are not valid in other functions.
  - The parameters of a function are also local variables that are only valid in the block of code of the function.

Scope of Variables

```
function volume_cylinder(radius, height)
  local volume
  volume = math.pi * (radius ^ 2) * height
  return volume
end


local radius, height, volume
io.write("Radius of the cylinder: ")
radius = io.read()
io.write("Height of the cylinder: ")
height = io.read()

volume = volume_cylinder(radius, height)

io.write("Volume of the cylinder: ", volume)
```

Variables with the same name, but with different scopes.

Scope of Variables

- Functions <u>receive values</u> as parameters and <u>return values</u> (not variables).

- **The name of the variables may be the same, but they are different variables**.

```
function doble_value(x)
  x = x * 2
  return x
end

local x = 5.0
io.write(doble_value(x))
io.write(x)
```

Output: 10.0

Output: 5.0

Exercises

1) Rewrite the first exercises using functions:
   a) Write a program that converts kilometers per hour to miles per hour (1 km/h is equal to 0.6213711922 mi/h).
   b) Write a program that calculates the perimeter of a rectangle ($P = 2(lenght + width)$).
   c) Write a program that takes hours and minutes as input, and calculates the total number of minutes.
   d) Write a program that takes minutes as input, and display the total number of hours and minutes.

Extra Exercises: http://www.inf.puc-rio.br/~elima/gameprog/ExtraExercisesLua.pdf (optional)