Lua

Created	@April 25, 2023 1:41 PM

Lua

History

- Created in 1993, Puc-Rio, Brazil
- Combines DEL and SOL into a single language
- SOL is sun in portuguese, Lua is moon in portuguese
- Originally designed for scientists and engineers
    - Simple syntax
- Lua 1.0 never released to the public
- Lua 1.1 released in July 1991 under a restricted liscence
- Lua 2.1 in 1995
    - “Fallbacks”
        - Indicated function is called if something bad happens
- Lua 2.4 in May 1996
    - External compiler
- Lua 3 in July 1997
    - tags and tag methods
- Lua 4 in November 2000
    - Preprocessor removed

  - Rewrote API
  - for loop added
- Lua 5 in April 2003
  - Metatables replace tags/tag methods
  - Booleans

Distinguishing Features/Application Domain

- Embedded - Can be added to an application
  - Good with libraries
- Meant to be use as an embedded language
- Doesn’t take up much space (compact file size)
- Scripting language (extending games)
- Good for database queries (Postgre SQL)

Primitives

- +, -, *, /
- Equality/Inequality
  - ‘==’ and ‘~=’
  - Assignment ‘=’
  - <, >, < =, > =
- Methods
  - Tables
    - Add values
    - insert
    - sort
    - etc.

- Strings
  - upper
  - lower
  - length
- Math
  - abs
  - ciel
  - floor
  - log10
  - randomseed
  - LOTS of built in math function!

Data Types

- Dynamically typed language
  - No need to declare type
- 8 data types
  - Nil
    - Basically null for lua
  - Boolean
  - Number
    - no ints or floats
      - Any number is a number
  - String
    - Array of characters
    - Can store unicode values (emojis)
  - Function

        - Blocks of reusable code
        - Can use C (its built in)
    - Userdata
        - malloc in C
        - holds data
        - Write code in C and the Lua API can send values via pointers
    - ~~Thread~~
        - Small unit of execution
        - Has built in functions
        - Threads run concurrently
    - Table

      `array = {}`

        - Index origin is 1
        - Mixed types allowed
        - Basically a dictionary of key:value pairs
    - Abstract Data Types
        - Lua doesnt really support classes

Scoping and Parameter Passing

- All variables are global
    - Assign it to nil to delete it
- Local variable defined with keyword ‘local’
    - only accessible in the block of code its declared
- do…end structure for local variables
- Local variables override global variables of the same name

- Lexical Scoping
    - A function within a function can access local variables within the parent function
    - considered ‘upvalues’
- **Pass by Value**
    - Reference is passed??? Isnt this pass by reference
- Functions called without sending parameters auto fill to nil
- Additional parameters get ignored if they aren’t expected
- ellipse ‘…’ special operator
    - Arbitrary number of values in a table

I/O Functionality

- `print()` prints a statement with a newline
- `io.write()` prints the statement without a newline
- `io.read()` reads what the user inputted into the program
- `string.format()`
    - %s string
    - %d ints
    - %f floats
    - %c char ASCII value
    - %x hex
    - %c number
- Reading/Writing Files
    - `io.open(”filename”, “mode”)`
        - ‘r’
        - ‘w’

        - ‘a’ append mode
        - ‘r+’ read and write
        - ‘w+’ removes data from a file or creates a new file with read and write permissions
    - `io.close(”filename”)`
    - `print(file:read())`
        - prints the first line
    - `io.tempfile()` deleted file after program is done
    - `io.type(file)` returns type of file

Control Structures

- Multiple Assignment allowed
    - a, b = 1, 2 (a = 1, b = 2)
    - Swap also allowed (a, b = b, a)
- a, b, c = 0 (a = 0, b = nil, c = nil)
- ‘..’ operator concatanates strings
    - a = “hello ”..”rocky”
        - a = “hello rocky”
    - concatenating numbers are not numeric - they are casted to strings
- if statements normal
    - if condition then
        - stuff
    - elseif condition then
        - stuff
    - else

        - stuff
    - end ← explicit terminator
- While loop is ended using end keyword
- For loops
    - Generic
        - Traverses using an iterator
        - pairs for ordered data, ipairs for unordered data
    - Numeric
        - Run for specified number of iterations
            - for i=1, 10(ending value), 2(increment value) do
                - stuff
            - end
    - ‘break’
    - ‘return’
- Recursion is supported in Lua
- Proper tail recursion
    - memory efficient recursion

Data Structures

- Arrays/Hashmaps
    - Values other than nil
    - made using {}
    - no fixed size
    - key:value pairs
    - a[key] = “value”
        - adds “value” as value to key to table ‘a’

- 2D arrays
    - Array of an arrays
- Linked Lists
    - Singly linked
        - refernces to the first node
    - Doubly
        - References to first and last nodes of list
    - value attribute holds a data value
    - elements cannot be found by index
- Queue
    - Type of list with first-in first-out
    - Insertion & Deletion functio for basic implementation
    - Two indices for a longer queue for efficiency
- Stacks
    - Implemented using tables
    - last in first out
    - Matamethods can be used for insertion/deletion
- Metatables & Metamethods
    - getn() returns # of lement in tables
    - setn() sets size of an array
    - insert()
    - remove()
    - inheritance supported using these

Assignment

```
function calcArea(type)
  if type == 1
    print("Enter length:")
    length = io.read()
    print("Enter width:")
    width = io.read()
    return length*width
  elseif type == 2
    print("Enter base length:")
    base = io.read()
    print("Enter height:")
    height = io.read()
    return .5*base*height
  elseif type == 3
    print("Enter radius:")
    radius = io.read()
    return math.pi * radius * radius
  else
    print("Incorrect type entered.")
  end
end
```