Lua basics

By Kubsy

Hi and welcome to the basics of Lua where this page will teach you about basic programming theory in Lua straight on point so you can start making Lua scripts for TombEngine.

This page will teach you:

- Variables and data types
- Mathematical operations and maths library
- decision-making
- relational operators and logical operators
- Loops
- Tables (Arrays)
- Creating functions for Volume triggers
- Creating functions inside the script
- Programming readability (Naming Conventions and magic number)

If you would like to learn more in-depth and more complicated stuff about Lua check out here: https://www.lua.org/pil/contents.html

If you feel ready to create some interesting setups and modify them then check out this page: https://github.com/MontyTRC89/TombEngine/wiki/Lua-Script-Tutorials

so let's start 😃

Variables and data types

Variables are essential in programming, they act like containers which store some data that you have specified, variables can also be reused for different scenarios.

In Lua, you can define a `Local` and a `global` variable. Local variables will exist only in that block where you defined it and initialised for example:

```
i = 10
while i < 10
  local name = "Tombengine"
  print(name)
  i = i + 1
end
```

In the above example the variable `name` exists only in the while loop (while loops will be explained later) hence, you cannot use it anywhere else but only inside the while loop, **local variables must have `local` keyword before declaring the variable name**.

The `i` variable on the other hand is a global variable which can be used anywhere in the script, they can be declared by variable name and initialise it with a data that you want to store, so in the above example, **`i` will have a value of 10**.

In Lua, there are 7 types of data that you may need to use during scripting:

1. Strings:

- Strings are: characters, words, phrases or sentences which you can store, this is particularly useful if you want to have a dialogue and you want to store text dialogue in a variable.
- to declare a string you can either put double quotes **""** or single quotes **''**
    - `local LaraCurious = "Huh, what is this?"`
    - `local WernerAngry = 'Lara, what are you doing!!'`

2. Integers:

- Integers are **whole numbers** which can be negative or non-negative (positive) again, useful for certain scenarios such as storing: health or timer, but also very useful to remove magic numbers (more on them later)
- To declare a integer, just declare a variable and initialise with a number:
    - `local health = 100`
    - `local negativeNumber = -6`

3. Floating-point number (decimals):

- Floating-point numbers are **decimal point numbers**, which also can be positive and negative. Same as integers, they can be used for many scenarios.
- To declare a floating-point number, do the same as declaring and initialising with a integer:
    - `local Rotationx = 23.9`
    - `local negativeRotationy = -55.5`

4. Booleans:

- Boolean data type can have only 2 values: true or false. They are used to set a variable to be either true or false and use it in a particular scenario.
    - example:

```
isLaraKilled = false
if isLaraKilled = true
   print("What have you done!!!???")
end if
```

5. Tables:

- Tables are essentially an array, and an array is a list of values. Tables are quite complex and may not be needed but still worth an explanation will be explained in **Tables (Arrays)** section

6. Userdata:

- Userdata is a very essential data type and will be very common during TombEngine lua scripting. Userdata are **anything that is not built-in lua type** hence in TombEngine you will have functions and methods which are: `GetMoveableByName` or `Color.new` (explanation on these are in the Docs)

- Userdata can be used like this: `local raptor = TEN.Objects.GetMoveableByName("raptor1")` or `Color.new(255, 255, 255)`

  (pssst, you don't have to write `TEN.Objects.` if you write this in your lua file:

  ```
  local Util = require("Util")
  Util.ShortenTENCalls()
  ```

  `Util` variable will store another lua file which is "Util.lua" and gets its functions (that file must be present in you script folder!!!!!) which will then call the `ShortenTENcalls()` function so you can save your fingers and don't have to type `TEN.Objects.` 😃)

7. Functions:

- Functions are another data type which will be required if you want to use volume triggers within Tomb Editor.
- Functions are blocks of code which can be reused as many times as you like in the script and in as many volume triggers as you like. More information on functions in "Creating function for Volume triggers" and "Creating functions inside the script" sections.

Mathematical Operations and Maths library

In programming, you can do arithmetic operators to Integers and decimal-point numbers in order to do some calculation. In lua, there are 7 types of Arithmetic. Before explaining these Operators, let's define some variables:

```
local a = 10
local b = 5
```

now we can use these 2 variables to perform arithmetic:

1. Addition:

- You can add variables as long as they have numbers:

  ```
  local c = a + b
  print(c)

  Output: 15
  ```

- Explanation: since we declared and initialised 2 variables already (a = 10 and b = 5) we declared another variable (c) which will add these 2 variables together.

2. Subtraction:

- You can subtract variables as long as they have numbers:

```
local c = a - b
print(c)

Output: 5
```

- Explanation: This will subtract 2 variables to give you c = 5

3. Multiplication

- You can multiply variables as long as they have numbers:
    - note: in programming, in order to do multiplication, you have to put an asterisk `*` not like in normal maths you put "x"

```
local c = a * b
print(c)

Output: 50
```

    - Explanation This will multiply 2 variables to give you c = 50

4. Division

- You can divide variables as long as they have numbers:
    - Note: in programming, you need to put `/` to divide.

```
local c = a / b
print(c)

Output: 2
```

    - Explanation: Here we have divided 2 variables to give you c = 2

5. Modulus

- A modulus (or a modulo) is a special operator which acts like division **but** it will return the **remainder**.
    - A modulus is performed with a `%` sign

```
local c = a % b
print(c)
```

```
Output: 0
```

- Explanation: since 10 divided by 5 is 2 then the remainder is 0 because it doesn't give you a "left over" number and 10 / 5 can be exactly divided which won't give you a fractional/decimal number
- if you were to put:

```
local a = 10
local b = 6
local c = a % b
print(c)

Output: 4
```

- then the remainder is 4 as 10 / 6 cannot be exactly divided and it's a decimal number.

6. Exponent (power operator)

- Exponent is a operator which will multiply the number by itself
  - An exponent is performed by: `a^b` where a = number and b = number of times to multiply itself by

```
local c = a^2
print(c)

Output: 100
```

  - Explanation: since a = 10 then 10 to the power of 2 is 100 (because 10 x 10 = 100)

7. Unary (negation)

- Unary is a special operator which will convert the positive number to a negative
  - Unary is performed by putting `-` to a variable

```
local c = -a
print(c)

Output: -10
```

  - explanation: we negated the `a` variable so now it became -10 (think of it as -1 x 10 or -(10) = -10)

All of these operators come in handy if you want to perform some sort of calculations in-game for example you can perform a tricky calculation to move a static or rotate it or even to check distances precisely and etc.

Lua mathematics library

In Lua, you have a mathematics library which lets you do even more complex mathematics, kind of similar to Scientific, Engineering mathematics. There is a full list of mathematical methods are explained in here: https://www.tutorialspoint.com/lua/lua_math_library.htm

(note:

- 1 radian = 57,3 degrees approx, 2π radians = 360 degrees, π radians = 180 degrees and so on...
- general formula: **to convert radians to degrees: degrees x (π/180) and from degrees to radians: Radians × (180/π)**)

Here's a list of some useful math methods you might use:

- `math.sin(x)` returns the sine of x (it is assumed that x is in radians)
    - `math.sin(π) output: 0 (this is because the sine graph cuts the x-axis at π hence 0`
    - `math.sin(π/3) output: 1/2 (exact value)`
    - `math.sin(29) output: -0.664 (to 3 significant figures)`
- `math.cos(x)` returns the cosine of x (assumed x is in radians)
    - `math.cos(π) output: -1 (minimum amplitude of cosine graph at π hence -1)`
    - `math.cos(15) output: -0.760 (3 significant figures)`
- `math.random (m, n)` randomises the number given the m and n **where m = minimum value and n - maximum value**
    - `math.random(1,7) output: returns a random number which is between 1 and 7`

Decision-making

In programming, Decision-making implies to making choices and evaluating if one condition is true or false, if either one of them is true then it will take that route however if it is false then it will take an alternative route.

in Lua there are 3 decision making statements:

- `if` statements
    - an if statement will check if the condition is true and if it is then it executes the code
    - ```
      local a = 2
      local b = 1
      local c = a + b

      if c == 3 then print("c is 3")

      OR

      if c == 3 then
      ```
      ```

```
   print("c is 3")
end
```

- In the above example, the if statements determines if c = 3 if it is then it prints that c contains 3 value. as you can see, you can make an inline if or a block if statement.

- `elseif` statement
  - elseif statement will trigger if the if statement appears to be false, you can add as many elseifs as you like
  - ```
    local a = 2
    local b = 2
    local c = a + b

    if c == 3 then
       print("c is 3")
    elseif c == 4 then
       print("c is 4")
    end
    ```

- `else` statement
  - else statement is used if neither if and elseif statement is true
  - ```
     local a = 1
     local b = 1
     local c = a + b

     if c == 3 then
        print("c is 3")
     elseif c == 4 then
        print("c is 4")
     else
        print("c is neither a 3 or a 4"
     end
    ```