GAME.DEV

Lua – part 1

BITS Pilani

Pilani Campus

A Basic Lua Program

innovate achieve lead

```
-- defines a factorial function
    function fact (n)
      if n == 0 then
        return 1
      else
        return n * fact(n-1)
      end
    end

    print("enter a number:")
    a = io.read("*number")        -- read a number
    print(fact(a))
```

Running the Lua program…

- Open Scite (installed in the lua directory)
- File -> Open the lua program
- Use f5 to run the program

Variables

- all are by default global
- prepend with keyword ‘local’ to keep local scope

- Identifiers in Lua can be any string of letters, digits, and underscores, not beginning with a digit
- Avoid identifiers starting with an underscore followed by one or more uppercase letters (e.g., _VERSION); they are reserved for special uses in Lua.
- Case-sensitive

- The following words are reserved; we cannot use them as identifiers:

```
and       break     do        else      elseif
end       false     for       function  if
in        local     nil       not       or
repeat    return    then      true      until
while
```

Datatypes

- There are eight basic types in Lua: *nil*, *boolean*, *number*, *string*, *userdata*, *function*, *thread*, and *table*

```
print(type("Hello world"))  --> string
print(type(10.4*3))         --> number
print(type(print))          --> function
print(type(type))           --> function
print(type(true))           --> boolean
print(type(nil))            --> nil
print(type(type(X)))        --> string
```

Datatypes cont’d

- num = 42 -- All numbers are doubles.
- s = ‘this is a string' -- Immutable strings like Python.
- t = "double-quotes are also fine"
- u = [[Double brackets start and end multi-line strings.]]
- t = **nil** -- Undefines t; Lua has garbage collection.

Comments


```
-- Two dashes start a one-line comment.


 --[[
    Adding two ['s and ]'s makes it a multi-line comment.
--]]
```

Arithmetic Operations

Lua supports the usual arithmetic operators:

- `+´ (addition),
- `-´ (subtraction),
- `*´ (multiplication),
- `/´ (division),
- `-´ (negation).
- `^´ (exponentiation)
- `%´ (modulo)

Relational & Logical Operators

Lua provides the following relational operators:

- < (less than)
- > (greater than)
- <= (less than or equal to)
- >= (greater than or equal to)
- == (is equal to)
- ~= (not equal to)

The logical operators are (small case):

- and
- or
- not

Conditionals

- An if statement tests its condition and executes its then-part or its else-part accordingly . The else-part is optional. Examples:

```
if a < 0 then
a = 0
end

if a < b then
return a
else
return b
end

if num> 40 then
print('over 40')
end
```

If…then…elseif

- To write nested ifs you can use elseif. It is similar to an else followed by an if, but it avoids the need for multiple ends:

```
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
```

Armstrong Numbers

An Armstrong number of three digits is an integer such that the sum of the cubes of its digits is equal to the number itself.

For example, 371 is an Armstrong number since 3**3 + 7**3 + 1**3 = 371.

Exercise 1.1

Given exercise file ex1.lua. It stores a 3 digit number in the variable num.

Check and print if the value stored in num is Armstrong.

Useful function:

x=23.5

y= math.floor(x) – value of y is 23

User Input

- word = **io.read()** --read uptill next whitespace as string
- number = **io.read(“*number”)** --read next number
- line= **io.read(“*line”)** --read next line as string
- all= **io.read(“*all”)** --read everything in input stream

- num = tonumber(word) --convert proper strings to numbers

Exercise 1.2

Modify the above program to take input from the user as the number to check for Armstrong property.

For loop

- This loop will execute something for each value of var from exp1 to exp2, using exp3 as the step to increment var.

```
for var = exp1, exp2, exp3 do
<something>
end
```

```
sum = 0
for i = 1, 100 do -- The range includes both ends.
sum = sum + i
end
```

break

- Combine with a conditional to finish a loop.
- Only breaks inner loop

```
for i = 1, 100 do -- The range includes both ends.
    if sum >= 20 then
    break
    end
sum = sum + i
end
```

Exercise 1.3

Find all Armstrong numbers in 3 digit numbers; i.e, from 100 to 999.

While/repeat-until loop


```
while num < 50 do
num = num + 1 -- No ++ or += type operators.
end

repeat
print('the way of the future')
num = num - 1
until num == 0
```

Exercise 1.4

Find all Armstrong numbers from 1 to 1000.

Functions

function function_name(arg_list)
<do_something>
<return_something>
end

```
function fact (n)
    ans = 1
    while n>0 do
        ans = ans * n
        n=n-1
    end
    return ans
end
```

return

- returns occasional results from a function
- finishes the function.
  - implicit return at the end of any function, so you do not need to write one if your function ends naturally , without returning any value.

- Can appear as the last statement in your chunk or just before an end, an else, or an until.

```
local i = 1
v=10
while true do
if i == v then return i end
i = i + 1
end
```

- If needed inside a block, use an explicit do block around the statement:

```
function foo ()
return --<< SYNTAX ERROR
-- 'return' is the last statement in the next block
do return end -- OK
<other statements>
end
```

Exercise 1.5

Write a function which returns a boolean value of true or false if the passed argument is an Armstrong number.

```
function isArmstrong(num)
    --[[
            return true if num is Armstrong
    --]]
end
```

Narcissistic Numbers

- Is a number that is the sum of its own digits each raised to the power of the number of digits.
- Eg: 8208 = 8^4 + 2^4 + + 0^4 + 8^4

Exercise of the Class.

A program to find all narcissistic numbers up to 7 digits long.

1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407, 1634, 8208, 9474, 54748, 92727, 93084, 548834, 1741725, 4210818, 9800817, 9926315