Lua Programming Cheat Sheet PAGE 1 of 2 BY SETH KENLON

Lua is a lightweight, fast, embeddable, dynamically typed language implemented as a C library. Lua code is executed by the ``lua`` bytecode interpreter.
For full documentation, see lua.org/manual

Lua Packages

Instead of `#include` (as in C), Lua uses `require` to include a library.

```
require("example")
-- this is a comment
```

Set `package.path` to the location of your bundled libraries. This appends `./lib` to the default path:

```
package.path = package.path .. ';lib/?.lua'
require("example")
```

Global variables

Lua is dynamically typed.

```
myint = 12
myfloat = myint+3.1415
a = "my "
b = a .. "string"
```

Local variables

All variables are global unless declared to be local:

```
function myvar()
 local var = 13
 return var
end
```

Lua Functions

Create a function with the keyword `function`. Terminate a function definition with the keyword `end`.

```
function myfunc(arg)
 local var = 13
 local total = var+arg
 return total
end

result = myfunc(29)
```

Lua data types

nil always means *nil* (nothing)
boolean either `false` or `true`
number an integer or a float
string any 8-bit value, including embedded zeros ('\0')
function may be either Lua or C code
userdata a block of raw memory for arbitrary C data
thread a coroutine (managed by Lua, not the OS)
table is an associative array

While loops

```
i = 0
while i < 10 do
 print("hello")
 i = i+1
end
```

If statements

```
if n == 99 then
  print(n)
elseif n == 98 then
  print(n+1)
else
  print("n is " .. n)
end
```

Tables

Lua's data-structuring mechanism, tables can represent arrays, lists, symbol tables, sets, records, graphs, trees, and can even mimic classes (with metatables.)

```
mytab = {"heart","diamond","spade","club"}
myheart = print(mytab[1])
```

Iterating over tables

```
for v in pairs(mytable) do
  print(mytable[v])
end
```

```
for index,value in ipairs(mytable) do
  print(index,value)
end
```

Metatables

Every value in Lua can have a *metatable*. A metatable defines the behavior of the original value under certain operations.

```
Card = { }

function Card.init(suit,value)
      local self = setmetatable({}, Card)
      self.suit = suit
      self.value = value
      return self
end
```

You can use metatables to serve the same purpose as a class in an object-oriented language.

```
mycard = Card("spade",1)
print(mycard.suit) -- prints "spade"
print(mycard.value) -- prints "1"
```

Interactive

The `lua` command fuatures an interactive command-line.

```
$ lua
Lua X.Y.Z Copyright (C) 1994-20XX Lua.org, PUC-Rio
> seed = math.randomseed(os.time())
> print(math.random(1,20))
20
> os.exit()
```

Math and logic

+ addition	& bitwise AND	== equal
- subtraction	| bitwise OR	~= not equal
* multiplication	~ bitwise exclusive OR	< less than
/ float division	>> right shift	> greater than
// floor division	<< left shift	<= less or equal
% modulo	~ unary bitwise NOT	>= greater or equal
^ exponentiation		