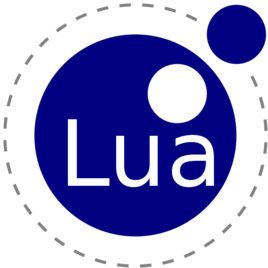
Lua教程

参考：<http://www.runoob.com/lua/lua-tutorial.html>

# Lua 教程



Lua 是一种轻量小巧的脚本语言，用标准C语言编写并以源代码形式开放， 其设计目的是为了嵌入应用程序中，从而为应用程序提供灵活的扩展和定制功能。

Lua 是巴西里约热内卢天主教大学（Pontifical Catholic University of Rio de Janeiro）里的一个研究小组，由Roberto Ierusalimschy、Waldemar Celes 和 Luiz Henrique de Figueiredo所组成并于1993年开发。

## 设计目的

其设计目的是为了嵌入应用程序中，从而为应用程序提供灵活的扩展和定制功能。

## Lua 特性

* **轻量级**: 它用标准C语言编写并以源代码形式开放，编译后仅仅一百余K，可以很方便的嵌入别的程序里。
* **可扩展**: Lua提供了非常易于使用的扩展接口和机制：由宿主语言(通常是C或C++)提供这些功能，Lua可以使用它们，就像是本来就内置的功能一样。
* **其它特性**:
  + 支持面向过程(procedure-oriented)编程和函数式编程(functional programming)；
  + 自动内存管理；只提供了一种通用类型的表（table），用它可以实现数组，哈希表，集合，对象；
  + 语言内置模式匹配；闭包(closure)；函数也可以看做一个值；提供多线程（协同进程，并非操作系统所支持的线程）支持；
  + 通过闭包和table可以很方便地支持面向对象编程所需要的一些关键机制，比如数据抽象，虚函数，继承和重载等。

## Lua 应用场景

* 游戏开发
* 独立应用脚本
* Web 应用脚本
* 扩展和数据库插件如：MySQL Proxy 和 MySQL WorkBench
* 安全系统，如入侵检测系统

## 第一个 Lua 程序

接下来我们使用 Lua 来输出"Hello World!"

## 实例(Lua 5.3)

print("Hello World!")

[**运行实例 »**](http://www.runoob.com/try/runcode.php?filename=HelloWorld&type=lua)

运行后，会在屏幕上显示 Hello, world!。

# Lua 环境安装

## Linux 系统上安装

Linux & Mac上安装 Lua 安装非常简单，只需要下载源码包并在终端解压编译即可，本文使用了5.3.0版本进行安装：

curl -R -O http://www.lua.org/ftp/lua-5.3.0.tar.gz

tar zxf lua-5.3.0.tar.gz

cd lua-5.3.0

make linux test

make install

## Mac OS X 系统上安装

curl -R -O http://www.lua.org/ftp/lua-5.3.0.tar.gz

tar zxf lua-5.3.0.tar.gz

cd lua-5.3.0

make macosx test

make install

接下来我们创建一个 HelloWorld.lua 文件，代码如下:

print("Hello World!")

执行以下命令:

$ lua HelloWorld.lua

输出结果为：

Hello World!

## Window 系统上安装 Lua

window下你可以使用一个叫"SciTE"的IDE环境来执行lua程序，下载地址为：

* 本站下载地址：[LuaForWindows\_v5.1.4-46.exe](http://static.runoob.com/download/LuaForWindows_v5.1.4-46.exe)
* Github 下载地址：<https://github.com/rjpcomputing/luaforwindows/releases>
* Google Code下载地址 : <https://code.google.com/p/luaforwindows/downloads/list>

双击安装后即可在该环境下编写 Lua 程序并运行。

你也可以使用 Lua 官方推荐的方法使用 LuaDist：[http://luadist.org/](http://luadist.org/" \t "_blank)

# Lua 基本语法

Lua 学习起来非常简单，我们可以创建第一个 Lua 程序！

## 第一个 Lua 程序

### 交互式编程

Lua 提供了交互式编程模式。我们可以在命令行中输入程序并立即查看效果。

Lua 交互式编程模式可以通过命令 lua -i 或 lua 来启用：

$ lua -i

$ Lua 5.3.0 Copyright (C) 1994-2015 Lua.org, PUC-Rio

>

在命令行中，输入以下命令:

> print("Hello World！") //输出hello world

接着我们按下回车键，输出结果如下：

> print("Hello World！")

Hello World！

>

D:\lua5.1>lua

Lua 5.1.4 Copyright (C) 1994-2008 Lua.org, PUC-Rio

> print hello world

stdin:1: '=' expected near 'hello'

> print("hello world")

hello world

> print("hello world!")

hello world!

## 脚本式编程

我们可以将 Lua 程序代码保持到一个以 lua 结尾的文件，并执行，该模式称为脚本式编程，如我们将如下代码存储在名为 hello.lua 的脚本文件中：

print("Hello World！")

print("www.w3cschool.cc")

使用 lua 名执行以上脚本，输出结果为：

$ lua test.lua

Hello World！

www.w3cschool.cc

我们也可以将代码修改为如下形式来执行脚本（在开头添加：#!/usr/local/bin/lua）：

#!/usr/local/bin/lua

print("Hello World！")

print("www.w3cschool.cc")

以上代码中，我们指定了 Lua 的解释器 /usr/local/bin directory。加上 # 号标记解释器会忽略它。接下来我们为脚本添加可执行权限，并执行：

./test.lua

Hello World！

www.w3cschool.cc

F:\Html\mannual\_files\_all\lua\_study\codes>lua hello.lua

hello world!

## 注释

### 单行注释

两个减号是单行注释:

--

### 多行注释

--[[

多行注释

多行注释

--]]

## 标示符

Lua 表示符用于定义一个变量，函数获取其他用户定义的项。标示符以一个字母 A 到 Z 或 a 到 z 或下划线 \_ 开头后加上0个或多个字母，下划线，数字（0到9）。

最好不要使用下划线加大写字母的标示符，因为Lua的保留字也是这样的。

Lua 不允许使用特殊字符如 @, $, 和 % 来定义标示符。 Lua 是一个区分大小写的编程语言。因此在 Lua 中 W3c 与 w3c 是两个不同的标示符。以下列出了一些正确的标示符：

mohd zara abc move\_name a\_123

myname50 \_temp j a23b9 retVal

## 关键词

以下列出了 Lua 的保留关键字。保留关键字不能作为常量或变量或其他用户自定义标示符：

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

一般约定，以下划线开头连接一串大写字母的名字（比如 \_VERSION）被保留用于 Lua 内部全局变量。

## 全局变量

在默认情况下，变量总是认为是全局的。

全局变量不需要声明，给一个变量赋值后即创建了这个全局变量，访问一个没有初始化的全局变量也不会出错，只不过得到的结果是：nil。

> print(b)

nil

> b=10

> print(b)

10

>

如果你想删除一个全局变量，只需要将变量赋值为nil。

b = nil

print(b) --> nil

这样变量b就好像从没被使用过一样。换句话说, 当且仅当一个变量不等于nil时，这个变量即存在。

# Lua 数据类型

Lua是动态类型语言，变量不要类型定义,只需要为变量赋值。 值可以存储在变量中，作为参数传递或结果返回。

Lua中有8个基本类型分别为：nil、boolean、number、string、userdata、function、thread和table。

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| nil | 这个最简单，只有值nil属于该类，表示一个无效值（在条件表达式中相当于false）。 |
| boolean | 包含两个值：false和true。 |
| number | 表示双精度类型的实浮点数 |
| string | 字符串由一对双引号或单引号来表示 |
| function | 由 C 或 Lua 编写的函数 |
| userdata | 表示任意存储在变量中的C数据结构 |
| thread | 表示执行的独立线路，用于执行协同程序 |
| table | Lua 中的表（table）其实是一个"关联数组"（associative arrays），数组的索引可以是数字或者是字符串。在 Lua 里，table 的创建是通过"构造表达式"来完成，最简单构造表达式是{}，用来创建一个空表。 |

我们可以使用type函数测试给定变量或者值的类型：

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

F:\Html\mannual\_files\_all\lua\_study\codes>lua 3\_0.lua

hello world! string

12 number

false boolean

nil nil

function: 00A77290 function

## nil（空）

nil 类型表示一种没有任何有效值（未定义变量或者赋值为nil的变量），它只有一个值 -- nil，例如打印一个没有赋值的变量，便会输出一个 nil 值：

> print(type(a))

nil

>

对于全局变量和 table，nil 还有一个"删除"作用，给全局变量或者 table 表里的变量赋一个 nil 值，等同于把它们删掉（不能被访问了，即使访问了也是一个没有意义的值），执行下面代码就知：

tab1 = { key1 = "val1", key2 = "val2", "val3" }

for k, v in pairs(tab1) do

print(k .. " - " .. v)

end

Key1不要加双引号

tab1.key1 = nil

for k, v in pairs(tab1) do

print(k .. " - " .. v)

end

print(tab1[“key1”]) //需要加双引号

print(tab1.key1) //不需要加双引号

## boolean（布尔）

boolean 类型只有两个可选值：true（真） 和 false（假），Lua 把 false 和 nil 看作是"假"，其他的都为"真":

print(type(true))

print(type(false))

print(type(nil))

if false or nil then

print("至少有一个是 true")

else

print("false 和 nil 都为 false!")

end

以上代码执行结果如下：

$ lua test.lua

boolean

boolean

nil

false 和 nil 都为 false!

## number（数字）

Lua 默认只有一种 number 类型 -- double（双精度）类型（默认类型可以修改 luaconf.h 里的定义），以下几种写法都被看作是 number 类型：

print(type(2))

print(type(2.2))

print(type(0.2))

print(type(2e+1))

print(type(0.2e-1))

print(type(7.8263692594256e-06))

print(2,type(2))

print(2.2,type(2.2))

print(0.1,type(0.1))

print(3e+2,type(3e+2))

print(2.13453223e+12,type(2.13453223e+12))

[**运行实例 »**](http://www.runoob.com/try/runcode.php?filename=datatype1&type=lua)

以上代码执行结果：

number

number

number

number

number

number

## string（字符串）

字符串由一对双引号或单引号来表示。

string1 = "this is string1"

string2 = 'this is string2'

也可以用 2 个方括号 "[[]]" 来表示"一块"字符串（类似PHP中的<<<eof eof）。

html = [[

<html>

<head></head>

<body>

<a href="http://www.w3cschool.cc/">w3cschool菜鸟教程</a>

</body>

</html>

]]

print(html)

以下代码执行结果为：

<html>

<head></head>

<body>

<a href="http://www.w3cschool.cc/">w3cschool菜鸟教程</a>

</body>

</html>

在对一个数字字符串上进行算术操作时，Lua 会尝试将这个数字字符串转成一个数字:

> print("2" + 6)

8.0

> print("2" + "6")

8.0

> print("2 + 6")

2 + 6

> print("-2e2" \* "6")

-1200.0

> print("error" + 1)

stdin:1: attempt to perform arithmetic on a string value

stack traceback:

stdin:1: in main chunk

[C]: in ?

>

以上代码中"error" + 1执行报错了，字符串连接使用的是 .. ，如：

> print("a" .. 'b')

ab

> print(157 .. 428)

157428

>

使用 # 来计算字符串的长度，放在字符串前面，如下实例：

> len = "www.w3cschool.cc"

> print(#len)

16

> print(#"www.w3cschool.cc")

16

>

## table（表）

在 Lua 里，table 的创建是通过"构造表达式"来完成，最简单构造表达式是{}，用来创建一个空表。也可以在表里添加一些数据，直接初始化表:

-- 创建一个空的 table

local tbl1 = {}

-- 直接初始表

local tbl2 = {"apple", "pear", "orange", "grape"}

Lua 中的表（table）其实是一个"关联数组"（associative arrays），数组的索引可以是数字或者是字符串。

-- table\_test.lua 脚本文件

a = {}

a["key"] = "value"

key = 10

a[key] = 22

a[key] = a[key] + 11

for k, v in pairs(a) do

print(k .. " : " .. v)

end

脚本执行结果为：

$ lua table\_test.lua

key : value

10 : 33

不同于其他语言的数组把 0 作为数组的初始索引，在 Lua 里表的默认初始索引一般以 1 开始。

-- table\_test2.lua 脚本文件

local tbl = {"apple", "pear", "orange", "grape"}

for key, val in pairs(tbl) do

print("Key", key)

end

脚本执行结果为：

$ lua table\_test2.lua

Key 1

Key 2

Key 3

Key 4

table 不会固定长度大小，有新数据添加时 table 长度会自动增长，没初始的 table 都是 nil。

-- table\_test3.lua 脚本文件

a3 = {}

for i = 1, 10 do

a3[i] = i

end

a3["key"] = "val"

print(a3["key"])

print(a3["none"])

脚本执行结果为：

$ lua table\_test3.lua

val

nil

## function（函数）

在 Lua 中，函数是被看作是"第一类值（First-Class Value）"，函数可以存在变量里:

-- function\_test.lua 脚本文件

function factorial1(n)

if n == 0 then

return 1

else

return n \* factorial1(n - 1)

end

end

print(factorial1(5))

factorial2 = factorial1

print(factorial2(5))

脚本执行结果为：

$ lua function\_test.lua

120

120

function 可以以匿名函数（anonymous function）的方式通过参数传递:

-- function\_test2.lua 脚本文件

function anonymous(tab, fun)

for k, v in pairs(tab) do

print(fun(k, v))

end

end

tab = { key1 = "val1", key2 = "val2" }

anonymous(tab, function(key, val)

return key .. " = " .. val

end)

脚本执行结果为：

$ lua function\_test2.lua

key1 = val1

key2 = val2

## thread（线程）

在 Lua 里，最主要的线程是协同程序（coroutine）。它跟线程（thread）差不多，拥有自己独立的栈、局部变量和指令指针，可以跟其他协同程序共享全局变量和其他大部分东西。

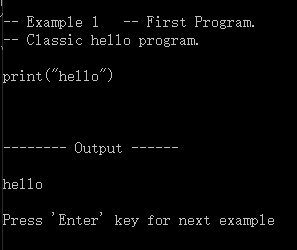
线程跟协程的区别：线程可以同时多个运行，而协程任意时刻只能运行一个，并且处于运行状态的协程只有被挂起（suspend）时才会暂停。

## userdata（自定义类型）

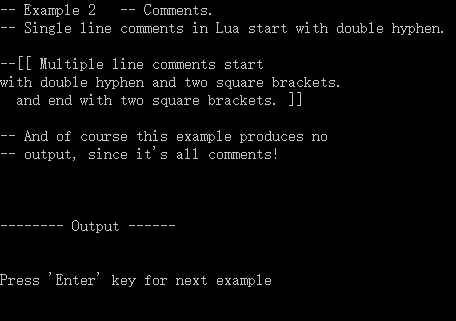
userdata 是一种用户自定义数据，用于表示一种由应用程序或 C/C++ 语言库所创建的类型，可以将任意 C/C++ 的任意数据类型的数据（通常是 struct 和 指针）存储到 Lua 变量中调用。

Luaforwindows安装后示例：

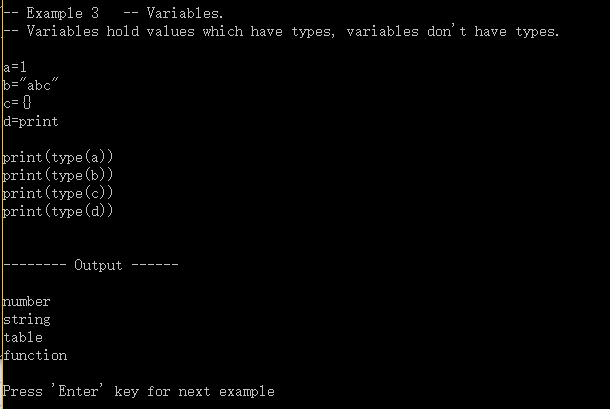
Eg.1



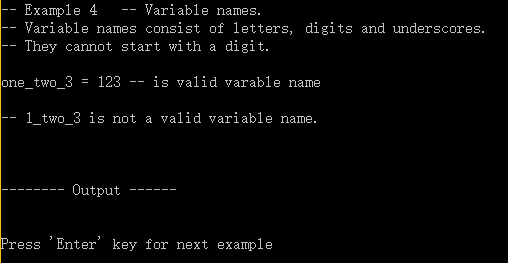
Eg.2



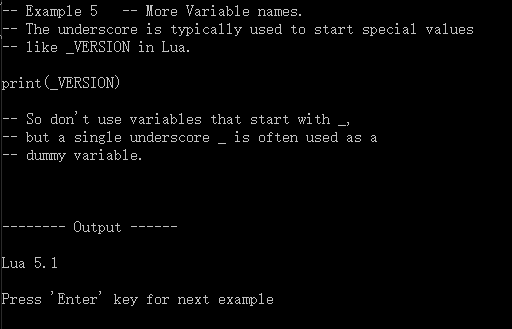
Eg.3



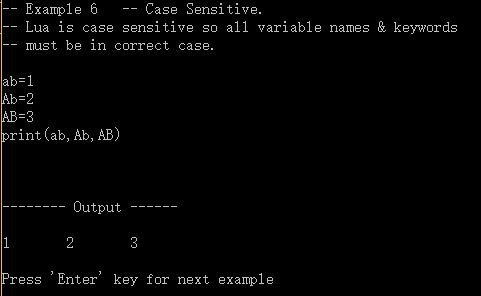
Eg.4



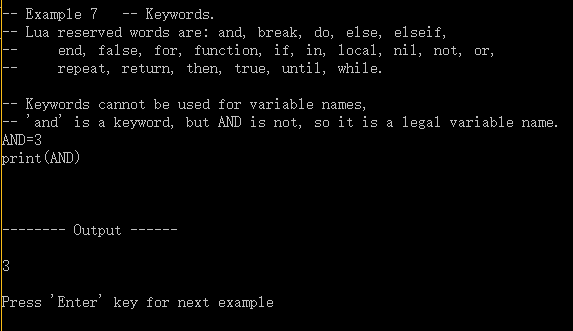
Eg.5



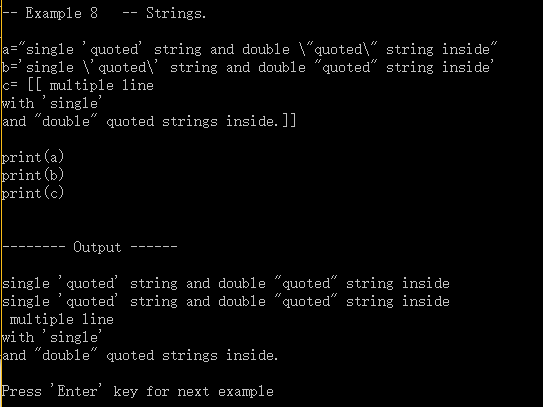
Eg.6



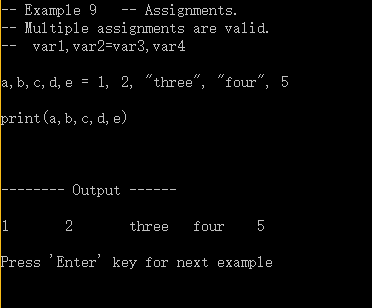
Eg.7



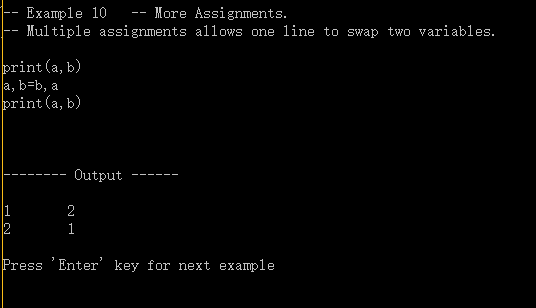
Eg.8



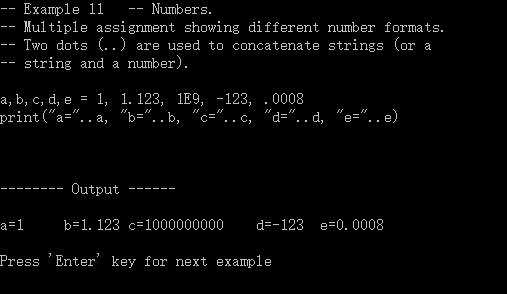
Eg.9



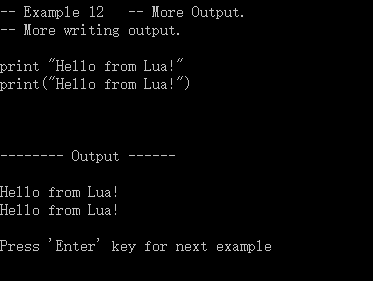
Eg.10



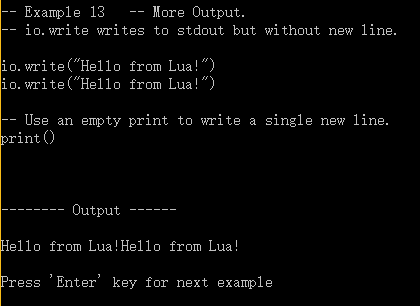
Eg.11



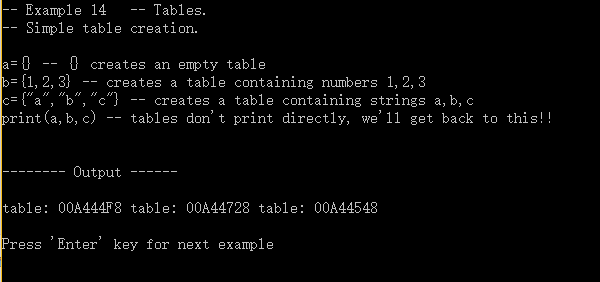
Eg.11



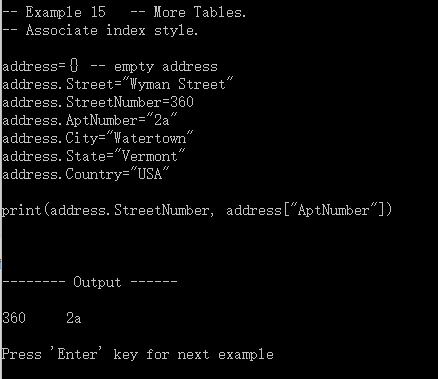
Eg.12



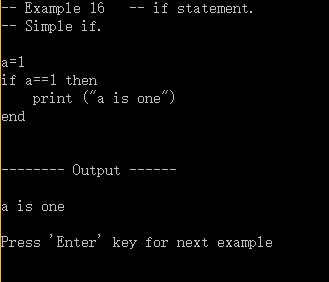
Eg.14



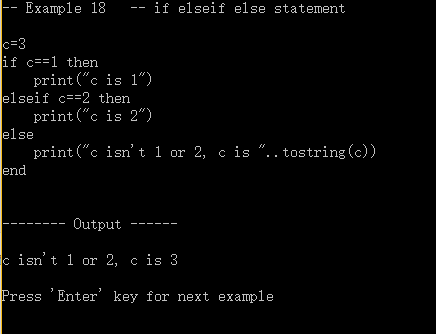
Eg.15



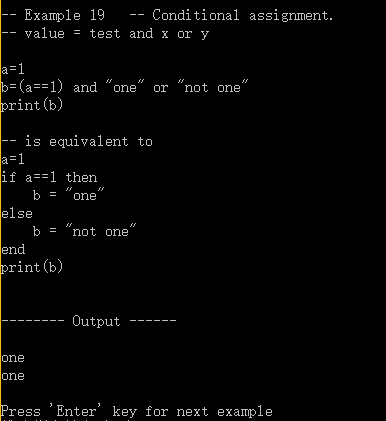
Eg.16



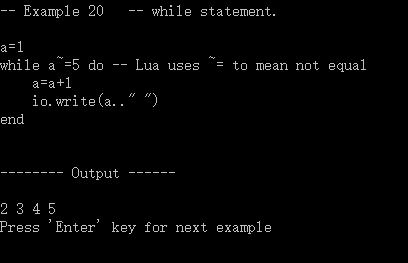
Eg.18



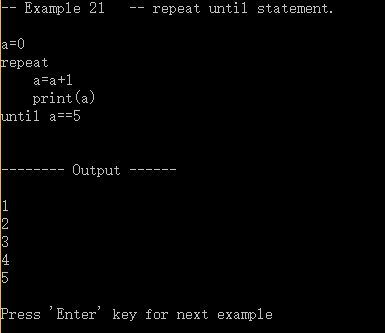
Eg.19



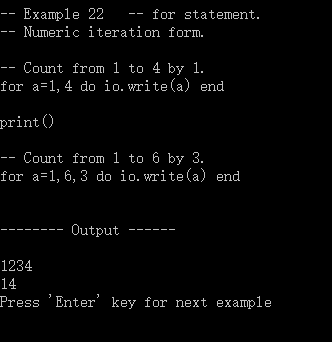
Eg.20



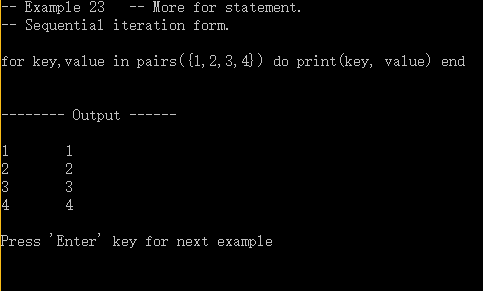
Eg.21



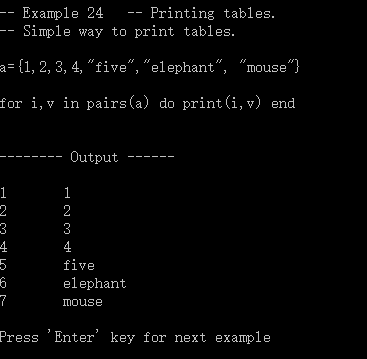
Eg.22



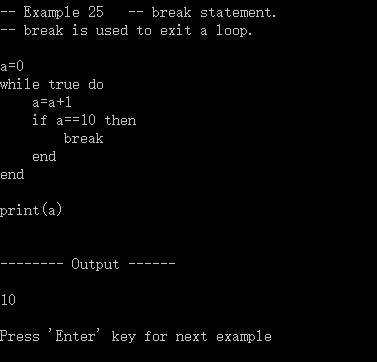
Eg.23



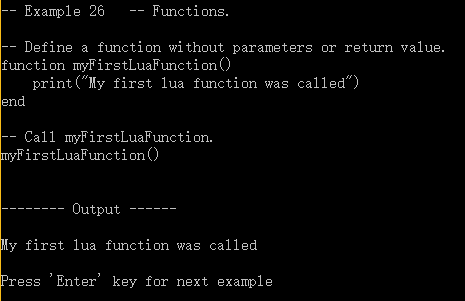
Eg.24



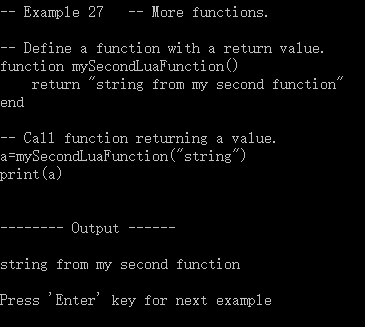
Eg.25



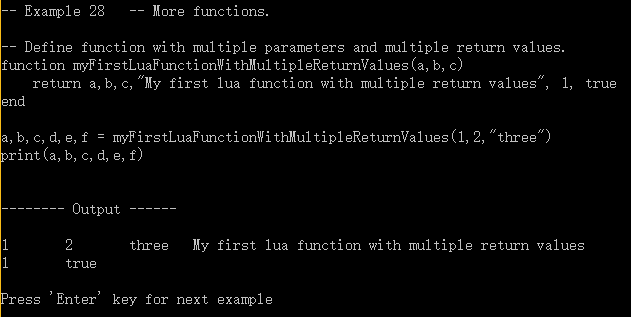
Eg.26



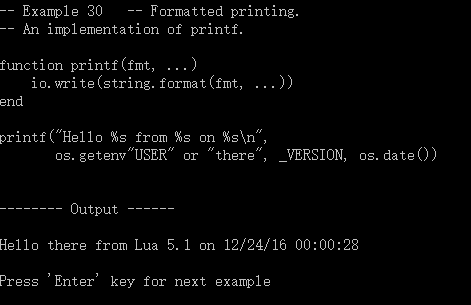
Eg.27



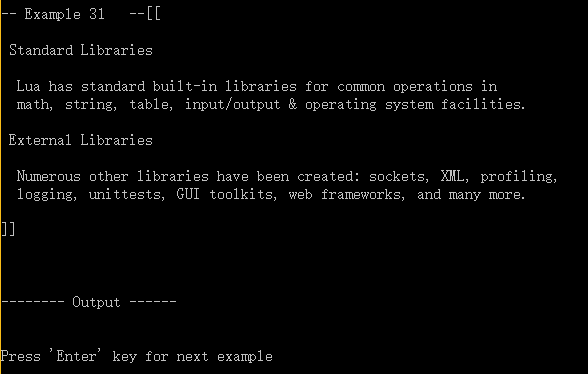
Eg.28



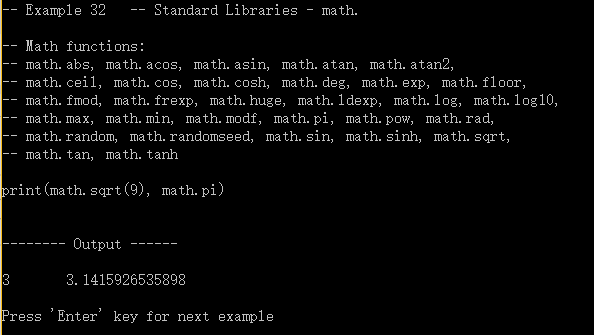
Eg.30



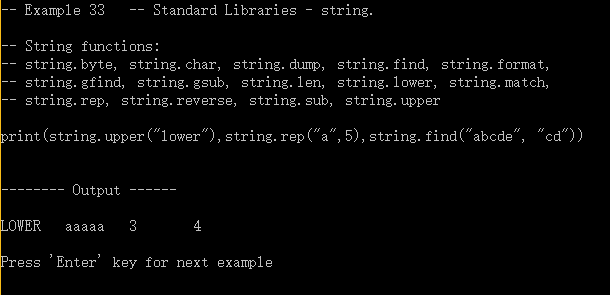
Eg.31



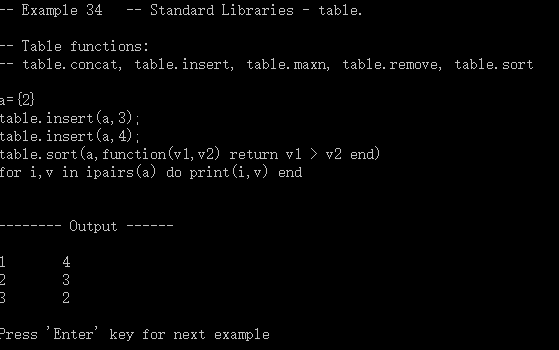
Eg.32



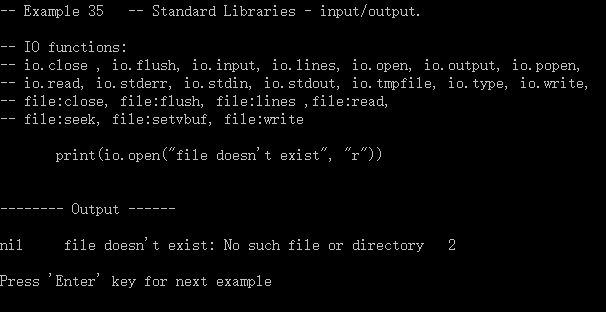
Eg.33



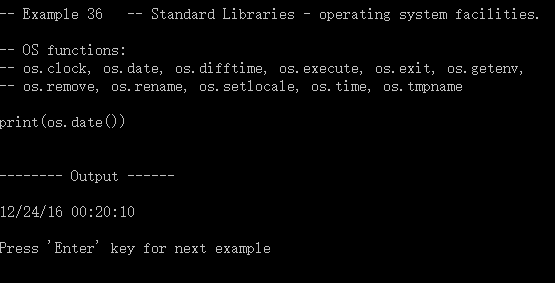
Eg.34



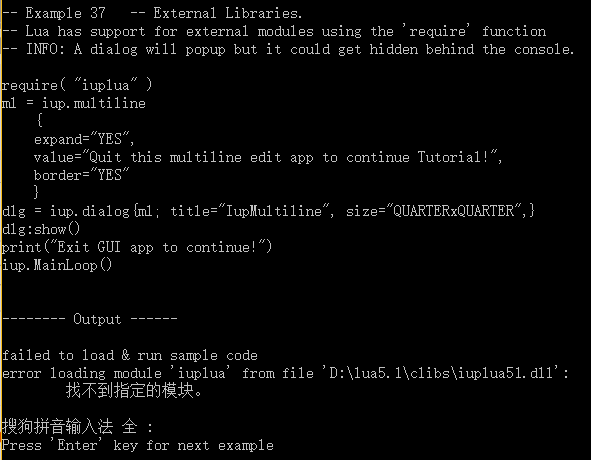
Eg.35



Eg.36



Eg.37



Eg.38

