函数

函数作用简单来看可有二

- 完成指定任务
- 来计算并返回数值

语法

```
function 函数名（参数表）
   执行部分
end;
```

即使参数表为空，也要使用 () 符号表示函数调用

但上述的 () 规则遇到如下两种情况是可以打破的(但是还是尽量正常写吧)这两种情况 () 是可选的

- 函数只有一个参数，且参数为字符串
- 函数只有一个参数，且参数为table

e.g

```
print "hello"                --等同于print("hello")
function_a {x = 10, y = 20} --等同于function_a({x = 10, y = 20})
```

面向对象式调用

在table表中引入函数作为成员时，可用

```
table1.function1(……)
table2:function2(……) --这二者是等价的调用方式
```

编写特性

*Lua*的函数可以使用其他语言进行编写不一定要使用*lua*来进行编写

参数匹配

Lua的函数的参数表赋值与Lua的赋值匹配相同，多余被忽略，缺少的以nil值进行补足

```
function f(a , b)
   return a or b
end

f(3)              --传入的值为(3,nil)
f(3 , 4)          --传入的值为(3,4)
f(3 , 4 , 5)      --传入的值为(3,4)
```

返回多个值

Lua的函数可以返回多个值，并且不限种类，可以混合

```
function a(……)
   ……
```

```
    return 变量1,变量2.....
end
```

在接收时，注意要使用多个变量接收，即使是table变量如果不加申明(标明索引一个一个写)，如果是空表多余量被忽略，只接收第一个值且表退化为变量，如果是非空只接报错

```
local function f3(a,b,c)
    return a,b,c
end

local w = {}
w = f3(1,"Spring Festival",false) --只接收第一个返回值，并且是空表的话退化到变量
local w1 = {1, 3, 4}
w1 = f3(1,"Spring Festival",false)--报错
local w2 = {}
w2[1],w2[2],w2[6] = f3(1,"Spring Festival",false) --正确，都收到了
```

三种不同返回值情况

```
function f00() end                  --返回nil
function f01() return "a" end       --返回一个值
function f02() return "q","w" end --返回多个值

--如果接收变量数 > 返回值数量，则多余部分都为nil
```

函数返回值和其他值混合赋予

- 当函数作为最后一个值或唯一一个值时，返回所有返回值

  ```
  function f03() return "a","b" end
  x,y = f03()      --x="a",y="b"返回完全
  x,y,z = 20,f03()--x=20,y="a",z="b"返回完全
  ```

- 当函数不为最后一个时，只返回第一个值

  ```
  function f03() return "a","b" end
  x,y = f03(),20  --x="a",y=20返回不完全
  ```

- 以上调用规则在直接调用作为函数参数时依然有效

```
function f03() return "a","b" end
print(f03())    --全返回全使用作为函数参数
print(20,f03()) --全返回全使用作为函数参数
print(f03(),20) --不全返回全使用作为函数参数，只返回第一个
print(fo3()..20)--不全返回全使用作为函数参数，只返回第一个
```

- 返回值作为初始化表参数时以上一二规则依然有效

```
function f03() return "a","b" end
local w = {f03()}    --全返回作为参数
local w2 = {20,f03()}--全返回作为参数
local w3 = {f03(),20}--只返回第一个作为参数
```

- **return f()**调用会返回**f()**的返回值，返回规则同一二

```
function f03() return "a","b" end
function f04() return f03() end   --全返回
function f05() return 20,f03() end--全返回
function f04() return f03(),20 end--只返回第一个
```

- 函数外面加括号强制返回第一个

```
function f03() return "a","b" end
x,y = (f03())
print((f03()))  --以上两种情况返回都强制只返回第一个
```

unpack函数

返回数组中的所有值(不是table，自定义的带指代的都不可用，自定义的索引也不可用)

最后可用版本5.1

可变参数

Lua可以接收可变数目的参数

和C语言类似都是使用三点(...)表示函数有可变参数

... 会生成一个arg表，可变的参数就都收入arg表中

示例：重写print

```
local printResult = ""

local function print(...)
   for i,v in ipairs(arg) do
      printResult = printResult .. tostring(v) .. "\t"
   end
   printResult = printResult .. "\n"
end
```

固定几个参数再加可变参数

```
function g(a , b , ...)  --前两个参数都由a，b接收，之后的才由arg表接收
   ……
end
```

虚变量忽略写法类似于matlab当中~的写法

```
local _,x = f1()  --假设f1()有2个输出，但你只想要第二个
local _,_,z = f2()--假设f2()有3个输出，但你只想要第三个
--下划线_就是省略符，他们可以占用变量读取位，但并不保存他们
```

命名函数

是我们的函数调用一个固定格式的表，这个表中的变量用指代来确定

```
local function change(bian)
   bian.old,bian.new = bian.new,bian,old --一个交换函数这样写就显得简洁明了，但table也
要适配
end

local x = {old = "2023", new = "2020"}
change(x)
```

函数特性

- 在**Lua**中函数并不是形参，如果在函数内部改变了值，在外部它也改变了，**Lua**传的是一个指针

再论函数

Lua中的函数是带有此法界定(lexical scoping)的第一类值(first-class value)

第一类值

Lua中的函数和其他值**(**数值，字符串**)**一样，可以被存放在变量中，表中，甚至是作为函数的返回值，函数的参数