函数

- 代码的一种组织形式
- 一个函数一般完成一项特定的功能
- 函数需要定义
    - 函数需要定义
    - 使用函数,俗称调用

```
# 定义一个函数
# 只是定义不会执行
# def 使关键字,后加一个空格
# 函数名,自己定义,起名时不能用大驼峰
# 后面括号和冒号不能省略,函数内所有代码要缩进

def fun():
    print("hello world")
print("这句话不属于函数")
```

```
这句话不属于函数
```

```
#函数的调用
fun()
```

```
hello world
```

函数的参数和返回值

- 参数: 负责给函数传递一些必要的数据和信息
    - 形参(形式参数): 在函数定义的时候参数没有具体的值,只是一个占位符
    - 实参(实际参数): 在调用函数时输入的值

- 返回值: 函数的执行结果
    - 使用return关键字
    - 如果没有return,系统会默认返回none
    - 函数一旦执行return,**无条件**返回,即结束函数的执行

```
# 参数的定义与使用
# 参数person只是一个符号,代表的是调用的时候某一个数据.
# 调用的时候,会用P代替函数里的所有person
def helle(preson):
    print("{0},吃了吗?".format(preson))
p = "xiaoming"
helle(p)
```

```
xiaoming,吃了吗?
```

```
# return 语句的基本使用
def holle(preson):
    print("{0},吃了吗?".format(preson))
    return "{0}没有理我".format(preson)
p = "xiaoming"
str =holle(p)
print(str)
```

```
xiaoming,吃了吗?
xiaoming没有理我
```

```
# 查找函数帮助文档
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

```
# 练习 做一个99乘法表 version
for row in range(1,10):
    for col in range(1,row+1):
       print( row*col, end=" ")
    print("----------------------")
```

```
1 ---------------------
2 4 ---------------------
3 6 9 ---------------------
4 8 12 16 ---------------------
5 10 15 20 25 ---------------------
6 12 18 24 30 36 ---------------------
7 14 21 28 35 42 49 ---------------------
8 16 24 32 40 48 56 64 ---------------------
9 18 27 36 45 54 63 72 81 ---------------------
```

```
# 99乘法表 version 2
def printline(row):
    for col in range(1,row+1):
       print( row*col, end=" ")
    print("")

for row in range(1,10):
    printline(row)
```

```
1
2 4
3 6 9
4 8 12 16
5 10 15 20 25
6 12 18 24 30 36
7 14 21 28 35 42 49
8 16 24 32 40 48 56 64
9 18 27 36 45 54 63 72 81
```

参数详解

> *[参考资料]http://www.cnblogs.com/bingabcd/p/6671368.html *Python参考资料: headfirst Python

- 参数分类
  - 普通参数
  - 默认参数
  - 关键字参数
  - 搜集参数
- 普通参数
  - 参见上例子
  - 定义时直接定义变量名
  - 调用时将参数放入指定位置

def 函数名(参数一,参数二,.....): 函数体

调用 函数名 (参数一, 参数二,.....) # 调用时,具体参考参数的位置,按位置赋值

- 默认参数
  - 形参带有默认值
  - 调用时,如果没有对应的形参赋值,则使用默认值

  def func*name(p1=v1,p2=v2,.....): func*block

  #调用1 func_name()#不知道值

  #调用2#知道值 value1 = 100 value2 = 200 fun_name(value1,value2)

```
# 默认参数实例
def func_name(name,age,gender = "male"):
    if gender =="male":#这时的条件指令,一定要用比较符号,而非赋值
        print("他的名字是{0},年龄为{1}".format(name,age))
    else:
        print("她的名字是{0},年龄为{1}".format(name,age))
```

```
# 默认参数实例一
func_name("xiaoming",18)
```

```
他的名字是xiaoming,年龄为18
```

```
#默认参数实例二
func_name("xiaohong",16,"female")
```

```
她的名字是xiaohong,年龄为16
```

- 关键字参数

```
def func(p1=V2 p2=v2.......)
    func_body

调用参数:
func(p1=value1,p2=value2......)
```

  - 比较麻烦,但不容易混淆.一般实参和形参只是按照位置一一对应即可,容易出错.
  - 使用关键字参数,可以不考虑参数位置

```
# 关键字参数实例
def stu(name,age,addr):
    print("i am a student")
    print("我叫{0},我今年{1}岁了,我住{2}".format(name,age,addr))
n = "xiaoming"
a = 18
addr ="文昌街"
stu(n,a,addr)
# 但若参数的顺序出错,则全出错



def stu_key(name="no name",age=0,addr="no adder"):
    print("i am a student")
    print("我叫{0},我今年{1}岁了,我住{2}".format(name,age,addr))
n = "xiaoming"#在"="前后加一个空格
a = 18
addr = "文昌街"
stu_key(age=a, name=n, addr=addr)#在逗号后加一个空格
#这时不需要考虑顺序
```

```
i am a student
我叫xiaoming,我今年18岁了,我住文昌街
i am a student
我叫xiaoming,我今年18岁了,我住文昌街
```

- 收集参数
  - 把没有位置,不能和定义时的参数位置对应的参数,放入一个特定的数据结构中
  - 语法

    def func(*args): func_body 按照list使用方法访问args的传入参数

    调用: func(P1, p2.....)
  - 参数名args不是必须这么写,但是强烈推荐.
  - 参数名args前需要有*
  - 搜集参数可以与其他参数共存
  - 搜集参数的关键字
    - 把关键字参数按字典格式存放到搜集参数中

      def func(**kwargs): func_body

调用: func(p1=l1, p2=l2, p3=l3.....)

- kwargs是约定俗成
- 调用时把多余关键字参数存放到kwargs
- 访问kwargs需要字典格式访问

```
#搜集参数案例
#可以将args看做一个列表
def stu(*args):
    print(type(args))
#type用来检测变量类型
    for number in args:
        print(number)
stu(1, 2, 3, 4, 5, 6)
```

```
<class 'tuple'>
1
2
3
4
5
6
```

```
#搜集参数案例2
#收集参数可以不带任何实参
stu()
```

```
<class 'tuple'>
```

```
#搜集参数的关键字的案例
def stu(**kwargs):
    print(type(kwargs))
    #字典的访问
    for k,v in kwargs.items():
        print(k,v)


stu(qiqi=1, wiwi=2, eiei=3)
print("*********")
stu(qiqi=1)
```

```
<class 'dict'>
qiqi 1
wiwi 2
eiei 3
*********
<class 'dict'>
qiqi 1
```

- 收集参数混合使用的问题
  - 搜集函数,关键字参数和普通参数可以混合使用
  - 普通参数和关键字参数优先调用
  - 定义时一般先找普通参数,关键字参数,搜集参数tuple,搜集参数dict

```
def stu(name, age, hobby="no", *args, **kwargs):
    print("我叫{0},年龄{1}".format(name, age))
    if hobby =="no":
        print("我没有爱好")
    else:
        print("我的爱好是{0}".format(hobby))

    print("*"*20)

    for i in args:
        print(i)

    print("#"*20)

    for k,v in kwargs.items():
        print(k,">>>",v)

name ="xiaoming"
age ="19"
hobby="coding"
stu(name,age,hobby, 1, 2, 3, qiqi=4, wiwi=5,)
```

```
我叫xiaoming,年龄19
我的爱好是coding
********************
1
2
3
####################
qiqi >>> 4
wiwi >>> 5
```

返回值

- 函数和过程的区别
  - 有无返回值

- 需要return显示返回内容
- 如果没有返回,则默认返回None
- 推荐结束时加一个return

```
#返回值实例
def func_1():
    print("hello world")
    return 1
def func_2():
    print("hello world")

f1 = func_1()
print(f1)
f2 = func_2()
print(f2)
```

```
hello world
1
hello world
None
```

函数文档

- 函数的文档的作用是对当前函数提供相关的参考价值
- 文档的写法
  - 在函数内部开始的第一行使用**三引号**定义符
  - 一般具有固定格式

- 文档的查看
  - 使用help函数
  - 使用**.doc**

```
def func_1():
    '''
    这是文档
    一般用三引号
    '''
    print("hello world")
func_1()
help(func_1)
func_1.__doc__
```

```
hello world
Help on function func_1 in module __main__:

func_1()
    这是文档
    一般用三引号
```

```
'这是文档\n    一般用三引号'
```

变量作用域

- 变量的作用范围限制
- 分类
  - 全局(global):在函数外部定义
  - 局部(local):在函数内部定义

- 变量的作用范围:
  - 全局变量:在全局范围都有效
  - 全局变量在局部都可以使用(即函数内部都可以访问函数外部定义的变量)
  - 局部变量:在局部范围可以使用
  - 局部变量在全局范围不可使用

- LEGB原则:
  - L(local):局部作用域
  - E(enclosing function locale) 外部嵌套作用域
  - G(Global module) 函数定义所在的模块作用域
  - B(Buildin) python内置模块的作用域

```
a1=100#a1是全局的
def fun():
    print(a1)
    a2 = 99
    print(a2)
fun()
print(a2)#a2是局部变量,全局不能使用.
```

```
100
99



---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-6-cfe53667cccb> in <module>()
      5     print(a2)
      6 fun()
----> 7 print(a2)


NameError: name 'a2' is not defined
```

提升局部变量为全局变量

- 使用global

```
b1=100#a1是全局的
def fun():
    print(b1)
    global b2
    b2 = 99
    print(b2)
fun()
print(b2)#a2是局部变量,全局不能使用.
```

```
  File "<ipython-input-9-6c801a7bbeff>", line 4
    global b2 = 99
              ^
SyntaxError: invalid syntax
```

globals,locals函数

- 可以使用globals和locals显示出全局变量和局部变量

```
# globals 和locals函数实例,这些函数叫做内建函数
a=1
b=2

def fun():
    c=4
    print("locals={0}".format(locals()))
    print("globals={0}".format(globals()))

fun()
```

```
locals={'c': 4}
globals={'__name__': '__main__', '__doc__': 'Automatically created module for IPyt
hon interactive environment', '__package__': None, '__loader__': None, '__spec__':
 None, '__builtin__': <module 'builtins' (built-in)>, '__builtins__': <module 'bui
ltins' (built-in)>, '_ih': ['', 'a1=100#a1是全局的\ndef fun():\n    print(a1)\n    a
2 = 99\n    print(a2)\ndef(a1)\nprint(a2)', 'a1=100#a1是全局的\ndef fun():\n    prin
t(a1)\n    a2 = 99\n    print(a2)\ndef(a1)\n#print(a2)', 'a1=100#a1是全局的\ndef fun
():\n    print(a1)\n    a2 = 99\n    print(a2)\ndef(a1)\n#print(a2)', 'a1=100#a1是
全局的\ndef fun():\n    print(a1)\n    a2 = 99\n    print(a2)\nfun(a1)\n#print(a2)'
, 'a1=100#a1是全局的\ndef fun():\n    print(a1)\n    a2 = 99\n    print(a2)\nfun()\n
#print(a2)', 'a1=100#a1是全局的\ndef fun():\n    print(a1)\n    a2 = 99\n    print(a
2)\nfun()\nprint(a2)', 'b1=100#a1是全局的\ndef fun():\n    print(b1)\n    global a2
= 99\n    print(b2)\nfun()\nprint(a2)#a2是局部变量,全局不能使用.', 'b1=100#a1是全局的\n
def fun():\n    print(b1)\n    global b2 \n    b2 = 99\n    print(b2)\nfun()\nprin
t(b2)#a2是局部变量,全局不能使用.', 'b1=100#a1是全局的\ndef fun():\n    print(b1)\n    g
lobal b2 = 99 \n   # b2 = 99\n    print(b2)\nfun()\nprint(b2)#a2是局部变量,全局不能使
用.', '# globals 和locals函数实例\na=1\nb=2\n\ndef fun(c,d):\n    c=4\n    print("lo
cals={0}".format(locals()))\n    print("globals={0}".format(globals()))\n\nfun()',
 '# globals 和locals函数实例\na=1\nb=2\n\ndef fun():\n    c=4\n    print("locals={0}
".format(locals()))\n    print("globals={0}".format(globals()))\n\nfun()'], '_oh':
 {}, '_dh': ['/Users/zhangchengxin/Python学习'], 'In': ['', 'a1=100#a1是全局的\ndef
fun():\n    print(a1)\n    a2 = 99\n    print(a2)\ndef(a1)\nprint(a2)', 'a1=100#a1
是全局的\ndef fun():\n    print(a1)\n    a2 = 99\n    print(a2)\ndef(a1)\n#print(a2)
', 'a1=100#a1是全局的\ndef fun():\n    print(a1)\n    a2 = 99\n    print(a2)\ndef(a1
)\n#print(a2)', 'a1=100#a1是全局的\ndef fun():\n    print(a1)\n    a2 = 99\n    prin
t(a2)\nfun(a1)\n#print(a2)', 'a1=100#a1是全局的\ndef fun():\n    print(a1)\n    a2 =
 99\n    print(a2)\nfun()\n#print(a2)', 'a1=100#a1是全局的\ndef fun():\n    print(a1
)\n    a2 = 99\n    print(a2)\nfun()\nprint(a2)', 'b1=100#a1是全局的\ndef fun():\n
  print(b1)\n    global a2 = 99\n    print(b2)\nfun()\nprint(a2)#a2是局部变量,全局不
能使用.', 'b1=100#a1是全局的\ndef fun():\n    print(b1)\n    global b2 \n    b2 = 99\
n    print(b2)\nfun()\nprint(b2)#a2是局部变量,全局不能使用.', 'b1=100#a1是全局的\ndef f
un():\n    print(b1)\n    global b2 = 99 \n   # b2 = 99\n    print(b2)\nfun()\npri
nt(b2)#a2是局部变量,全局不能使用.', '# globals 和locals函数实例\na=1\nb=2\n\ndef fun(c,d
):\n    c=4\n    print("locals={0}".format(locals()))\n    print("globals={0}".for
```

mat(globals()))\n\nfun()', '# globals 和locals函数实例\na=1\nb=2\n\ndef fun():\n c=4\n print("locals={0}".format(locals()))\n print("globals={0}".format(globals()))\n\nfun()'], 'Out': {}, 'get_ipython': <bound method InteractiveShell.get_ipython of <ipykernel.zmqshell.ZMQInteractiveShell object at 0x10f0b80f0>>, 'exit': <IPython.core.autocall.ZMQExitAutocall object at 0x10f0fb7b8>, 'quit': <IPython.core.autocall.ZMQExitAutocall object at 0x10f0fb7b8>, '_': '', '__': '', '___': '', '_i': '# globals 和locals函数实例\na=1\nb=2\n\ndef fun(c,d):\n c=4\n print("locals={0}".format(locals()))\n print("globals={0}".format(globals()))\n\nfun()', '_ii': 'b1=100#a1是全局的\ndef fun():\n print(b1)\n global b2 = 99 \n # b2 = 99\n print(b2)\nfun()\nprint(b2)#a2是局部变量,全局不能使用.', '_iii': 'b1=100#a1是全局的\ndef fun():\n print(b1)\n global b2 \n b2 = 99\n print(b2)\nfun()\nprint(b2)#a2是局部变量,全局不能使用.', '_i1': 'a1=100#a1是全局的\ndef fun():\n print(a1)\n a2 = 99\n print(a2)\ndef(a1)\nprint(a2)', '_i2': 'a1=100#a1是全局的\ndef fun():\n print(a1)\n a2 = 99\n print(a2)\ndef(a1)\n#print(a2)', '_i3': 'a1=100#a1是全局的\ndef fun():\n print(a1)\n a2 = 99\n print(a2)\ndef(a1)\n#print(a2)', '_i4': 'a1=100#a1是全局的\ndef fun():\n print(a1)\n a2 = 99\n print(a2)\nfun(a1)\n#print(a2)', 'a1': 100, 'fun': <function fun at 0x10f27d598>, '_i5': 'a1=100#a1是全局的\ndef fun():\n print(a1)\n a2 = 99\n print(a2)\nfun()\n#print(a2)', '_i6': 'a1=100#a1是全局的\ndef fun():\n print(a1)\n a2 = 99\n print(a2)\nfun()\nprint(a2)', '_i7': 'b1=100#a1是全局的\ndef fun():\n print(b1)\n global a2 = 99\n print(b2)\nfun()\nprint(a2)#a2是局部变量,全局不能使用.', '_i8': 'b1=100#a1是全局的\ndef fun():\n print(b1)\n global b2 \n b2 = 99\n print(b2)\nfun()\nprint(b2)#a2是局部变量,全局不能使用.', 'b1': 100, 'b2': 99, '_i9': 'b1=100#a1是全局的\ndef fun():\n print(b1)\n global b2 = 99 \n # b2 = 99\n print(b2)\nfun()\nprint(b2)#a2是局部变量,全局不能使用.', '_i10': '# globals 和locals函数实例\na=1\nb=2\n\ndef fun(c,d):\n c=4\n print("locals={0}".format(locals()))\n print("globals={0}".format(globals()))\n\nfun()', 'a': 1, 'b': 2, '_i11': '# globals 和locals函数实例\na=1\nb=2\n\ndef fun():\n c=4\n print("locals={0}".format(locals()))\n print("globals={0}".format(globals()))\n\nfun()'}

eval()函数

- 把一个字符串当做函数执行
- 用法

  eval(string_code,globals=None,locals=None)

exec()函数

- 和eval()函数用法一致,不过没有返回值'
- 用法

  exec(string_code,globals=None,locals=None)

```
x = 100
y = 200
z = eval("x+y")
print(z)
```

```
300
```

```
x = 100
y = 200
z = exec("x+y")
print(z)
```

```
None
```

递归函数

- 函数直接或间接调用自身
- 优点:简洁,容易理解
- 缺点:对递归深度有限制,消耗资源大
- Python对递归深度有限制,超过限制报错
- 在写递归函数时,一定要写结束条件

```
x = 0
def fun():
    global x
    x += 1
    print(x)
    # 函数自己调用自己
    fun()

fun()
```

```
1
2
3
...
2965
2966


---------------------------------------------------------------------------

RecursionError                            Traceback (most recent call last)

<ipython-input-35-034fabf441c1> in <module>()
      7     fun()
      8
----> 9 fun()


<ipython-input-35-034fabf441c1> in fun()
      5     print(x)
      6     # 函数自己调用自己
----> 7     fun()
      8
      9 fun()


... last 1 frames repeated, from the frame below ...


<ipython-input-35-034fabf441c1> in fun()
      5     print(x)
      6     # 函数自己调用自己
----> 7     fun()
      8
      9 fun()


RecursionError: maximum recursion depth exceeded in comparison
```

```
# 斐波那契数列
# 数学表达式为: F(1)=1, F(2)=1,F(N)=F(n-1)+F(n)
def fib(n):#n表示第n个斐波那契数列的值
    if n == 1:
        return 1

    if n ==2:
        return 1

    return fib(n-1)+fib(n-2)

print(fib(10))
```

```
55
```