Yuan先生

随笔 - 1 文章 - 141 评论 - 143

Py西游攻关之函数

一 函数是什么?

函数一词来源于数学，但编程中的「函数」概念，与数学中的函数是有很大不同的，具体区别，我们后面会讲，编程中的函数在英文中也有很多不同的叫法。在BASIC中叫做subroutine(子过程或子程序)，在Pascal中叫做procedure(过程)和function，在C中只有function，在Java里面叫做method。

函数能提高应用的模块性，和代码的重复利用率。你已经知道Python提供了许多内建函数，比如print()。但你也可以自己创建函数，这被叫做用户自定义函数。

定义: 函数是指将一组语句的集合通过一个名字(函数名)封装起来，要想执行这个函数，只需调用其函数名即可

特性:

1.代码重用

2.保持一致性

3.可扩展性

二 函数的创建

2.1 格式:

Python 定义函数使用 def 关键字，一般格式如下:

```
def 函数名（参数列表）:
    函数体
```

```
def hello():
    print('hello')

hello()#调用
```

2.2 函数名的命名规则:

- 函数名必须以下划线或字母开头，可以包含任意字母、数字或下划线的组合。不能使用任何的标点符号;
- 函数名是区分大小写的。
- 函数名不能是保留字。

2.3 形参和实参

形参：形式参数，不是实际存在，是虚拟变量。在定义函数和函数体的时候使用形参，目的是在函数调用时接收实参（实参个数，类型应与实参一一对应）

实参：实际参数，调用函数时传给函数的参数，可以是常量，变量，表达式，函数，传给形参

区别：形参是虚拟的，不占用内存空间，.形参变量只有在被调用时才分配内存单元，实参是一个变量，占用内存空间，数据传送单向，实参传给形参，不能形参传给实参

```
import time
times=time.strftime('%Y--%m--%d')
def f(time):
    print('Now  time is : %s'%times)
f(times)
```

2.4 实例

实例1:

```
def show_shopping_car():
    saving=1000000
    shopping_car=[
```

```
        ('Mac',9000),
        ('kindle',800),
        ('tesla',100000),
        ('Python book',105),
    ]
    print('您已经购买的商品如下'.center(50,'*'))
    for i ,v in enumerate(shopping_car,1):
        print('\033[35;1m %s:  %s \033[0m'%(i,v))

    expense=0
    for i in shopping_car:
        expense+=i[1]
    print('\n\033[32;1m您的余额为 %s \033[0m'%(saving-expense))
show_shopping_car()
```

实例2:

现在我们就用一个例子来说明函数的三个特性:

函数的特性展示

三 函数的参数

- 必备参数
- 关键字参数
- 默认参数
- 不定长参数

必需的参数:

必需参数须以正确的顺序传入函数。调用时的数量必须和声明时的一样。

```
def f(name,age):

    print('I am %s,I am %d'%(name,age))

f('alex',18)
f('alvin',16)
```

关键字参数:

关键字参数和函数调用关系紧密，函数调用使用关键字参数来确定传入的参数值。使用关键字参数允许函数调用时参数的顺序与声明时不一致，因为 Python 解释器能够用参数名匹配参数值。

```
def f(name,age):

    print('I am %s,I am %d'%(name,age))

# f(16,'alvin') #报错
f(age=16,name='alvin')
```

缺省参数（默认参数）:

调用函数时，缺省参数的值如果没有传入，则被认为是默认值。下例会打印默认的age，如果age没有被传入:

```
def print_info(name,age,sex='male'):

    print('Name:%s'%name)
    print('age:%s'%age)
    print('Sex:%s'%sex)
    return

print_info('alex',18)
print_info('铁锤',40,'female')
```

不定长参数

你可能需要一个函数能处理比当初声明时更多的参数。这些参数叫做不定长参数，和上述2种参数不同，声明时不会命名。

```
# def add(x,y):
#     return x+y

def add(*tuples):
    sum=0
    for v in tuples:
        sum+=v

    return sum

print(add(1,4,6,9))
print(add(1,4,6,9,5))
```

加了星号（*）的变量名会存放所有未命名的变量参数。而加(**)的变量名会存放命名的变量参数

```
def print_info(**kwargs):

    print(kwargs)
    for i in kwargs:
        print('%s:%s'%(i,kwargs[i]))#根据参数可以打印任意相关信息了

    return

print_info(name='alex',age=18,sex='female',hobby='girl',nationality='Chinese',ability='Python')

############################位置

def print_info(name,*args,**kwargs):#def print_info(name,**kwargs,*args):报错

    print('Name:%s'%name)

    print('args:',args)
    print('kwargs:',kwargs)

    return

print_info('alex',18,hobby='girl',nationality='Chinese',ability='Python')
# print_info(hobby='girl','alex',18,nationality='Chinese',ability='Python')  #报错
#print_info('alex',hobby='girl',18,nationality='Chinese',ability='Python')   #报错
```

注意，还可以这样传参：

```
def f(*args):
    print(args)

f(*[1,2,5])

def f(**kargs):
    print(kargs)

f(**{'name':'alex'})
```

补充（高阶函数）：

高阶函数是至少满足下列一个条件的函数:

- 接受一个或多个函数作为输入
- 输出一个函数

```
def add(x,y,f):
    return f(x) + f(y)

res = add(3,-6,abs)
print(res)
###############
def foo():
    x=3
    def bar():
```

```
        return x
    return bar
```

四 函数的返回值

要想获取函数的执行结果，就可以用return语句把结果返回

注意:

1. 函数在执行过程中只要遇到return语句，就会停止执行并返回结果，so 也可以理解为 return 语句代表着函数的结束
2. 如果未在函数中指定return,那这个函数的返回值为None
3. return多个对象，解释器会把这多个对象组装成一个元组作为一个一个整体结果输出。

五 作用域

5.1 作用域介绍

python中的作用域分4种情况：

- L：local，局部作用域，即函数中定义的变量；
- E：enclosing，嵌套的父级函数的局部作用域，即包含此函数的上级函数的局部作用域，但不是全局的；
- G：globa，全局变量，就是模块级别定义的变量；
- B：built-in，系统固定模块里面的变量，比如int, bytearray等。 搜索变量的优先级顺序依次是：作用域局部>外层作用域>当前模块中的全局>python内置作用域，也就是LEGB。

```
x = int(2.9)  # int built-in

g_count = 0  # global
def outer():
    o_count = 1  # enclosing
    def inner():
        i_count = 2  # local
        print(o_count)
    # print(i_count) 找不到
    inner()
outer()

# print(o_count) #找不到
```

当然，local和enclosing是相对的，enclosing变量相对上层来说也是local。

5.2 作用域产生

在Python中，只有模块（module），类（class）以及函数（def、lambda）才会引入新的作用域，其它的代码块（如if、try、for等）是不会引入新的作用域的，如下代码：

```
if 2>1:
    x = 1
print(x)  # 1
```

这个是没有问题的，if并没有引入一个新的作用域，x仍处在当前作用域中，后面代码可以使用。

```
def test():
    x = 2
print(x) # NameError: name 'x2' is not defined
```

def、class、lambda是可以引入新作用域的。

5.3 变量的修改

```
#################
x=6
def f2():
    print(x)
    x=5
f2()

# 错误的原因在于print(x)时,解释器会在局部作用域找,会找到x=5(函数已经加载到内存),但x使用在声明前了,所以报
```

```
# local variable 'x' referenced before assignment.如何证明找到了x=5呢?简单:注释掉x=5,x=6
# 报错为:name 'x' is not defined
#同理
x=6
def f2():
    x+=1 #local variable 'x' referenced before assignment.
f2()
```

5.4 global关键字

当内部作用域想修改外部作用域的变量时，就要用到global和nonlocal关键字了，当修改的变量是在全局作用域（global作用域）上的，就要使用global先声明一下，代码如下：

```
count = 10
def outer():
    global count
    print(count)
    count = 100
    print(count)
outer()
#10
#100
```

5.5 nonlocal关键字

global关键字声明的变量必须在全局作用域上，不能嵌套作用域上，当要修改嵌套作用域（enclosing作用域，外层非全局作用域）中的变量怎么办呢，这时就需要nonlocal关键字了

```
def outer():
    count = 10
    def inner():
        nonlocal count
        count = 20
        print(count)
    inner()
    print(count)
outer()
#20
#20
```

5.6 小结

（1）变量查找顺序：LEGB，作用域局部>外层作用域>当前模块中的全局>python内置作用域；

（2）只有模块、类、及函数才能引入新作用域；

（3）对于一个变量，内部作用域先声明就会覆盖外部变量，不声明直接使用，就会使用外部作用域的变量；

（4）内部作用域要修改外部作用域变量的值时，全局变量要使用global关键字，嵌套作用域变量要使用nonlocal关键字。nonlocal是python3新增的关键字，有了这个 关键字，就能完美的实现闭包了。

六 递归函数

定义：在函数内部，可以调用其他函数。如果一个函数在内部调用自身本身，这个函数就是递归函数。

实例1(阶乘)

```
def factorial(n):

    result=n
    for i in range(1,n):
        result*=i

    return result

print(factorial(4))


#**********递归*********
def factorial_new(n):
```

```
14
15      if n==1:
16          return 1
17      return n*factorial_new(n-1)
18
19  print(factorial_new(3))
```

实例2(斐波那契数列)

```
1   def fibo(n):
2
3       before=0
4       after=1
5       for i in range(n-1):
6           ret=before+after
7           before=after
8           after=ret
9
10      return ret
11
12  print(fibo(3))
13
14  #**************递归**********************
15  def fibo_new(n):#n可以为零，数列有［0］
16
17      if n <= 1:
18          return n
19      return(fibo_new(n-1) + fibo_new(n-2))
20
21  print(fibo_new(3))
```

```
1   print(fibo_new(30000))#maximum recursion depth exceeded in comparison
```

递归函数的优点:　是定义简单，逻辑清晰。理论上，所有的递归函数都可以写成循环的方式，但循环的逻辑不如递归清晰。

递归特性:

1. 必须有一个明确的结束条件

2. 每次进入更深一层递归时，问题规模相比上次递归都应有所减少

3. 递归效率不高，递归层次过多会导致栈溢出(在计算机中，函数调用是通过栈（stack）这种数据结构实现的，每当进入一个函数调用，栈就会加一层栈帧，每当函数返　回，栈就会减一层栈帧。由于栈的大小不是无限的，所以，递归调用的次数过多，会导致栈溢出。)

七 内置函数 (Py3.5)

	Built-in Functions			
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()

	Built-in Functions			
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

py2内置函数：https://docs.python.org/3.5/library/functions.html#repr

重要的内置函数：

1 **filter(function, sequence)**

```
str = ['a', 'b','c', 'd']

def fun1(s):
    if s != 'a':
        return s


ret = filter(fun1, str)

print(list(ret))# ret是一个迭代器对象
```

对sequence中的item依次执行function(item)，将执行结果为True的item做成一个filter object的迭代器返回。可以看作是过滤函数。

2 **map(function, sequence)**

```
str = [1, 2,'a', 'b']

def fun2(s):

    return s + "alvin"

ret = map(fun2, str)

print(ret)      #  map object的迭代器
print(list(ret))#  ['aalvin', 'balvin', 'calvin', 'dalvin']
```

对sequence中的item依次执行function(item)，将执行结果组成一个map object迭代器返回.
map也支持多个sequence，这就要求function也支持相应数量的参数输入：

```
ef add(x,y):
    return x+y
print (list(map(add, range(10), range(10))))##[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

3 reduce(function, sequence, starting_value)

```
from functools import reduce

def add1(x,y):
    return x + y

print (reduce(add1, range(1, 101)))## 4950 （注：1+2+...+99）

print (reduce(add1, range(1, 101), 20))## 4970 （注：1+2+...+99+20）
```

对sequence中的item顺序迭代调用function，如果有starting_value，还可以作为初始值调用.

4 lambda

普通函数与匿名函数的对比：

```
#普通函数
def add(a,b):
    return a + b

```

```
print add(2,3)


#匿名函数
add = lambda a,b : a + b
print add(2,3)


#========输出===========
5
5
```

匿名函数的命名规则，用lamdba 关键字标识，冒号（：）左侧表示函数接收的参数（a,b），冒号（：）右侧表示函数的返回值（a+b）。

因为lamdba在创建时不需要命名，所以，叫匿名函数

八 函数式编程

学会了上面几个重要的函数后，我们就可以来聊一聊函数式编程到底是个什么鬼

一 概念（函数式编程）

函数式编程是一种编程范式，我们常见的编程范式有**命令式编程**（Imperative programming），**函数式编程**，常见的面向对象编程是也是一种命令式编程。

命令式编程是面向**计算机硬件**的抽象，有**变量**（对应着存储单元），**赋值语句**（获取，存储指令），**表达式**（内存引用和算术运算）和**控制语句**（跳转指令），一句话，命令式程序就是一个**冯诺依曼机**的**指令序列**。
而函数式编程是面向数学的抽象，将计算描述为一种**表达式求值**，一句话，函数式程序就是一个**表达式**。

函数式编程的本质

函数式编程中的**函数**这个术语不是指计算机中的函数，而是指数学中的函数，即自变量的映射。也就是说一个函数的值仅决定于函数参数的值，不依赖其他状态。比如y=x*x函数计算x的平方根，只要x的平方，不论什么时候调用，调用几次，值都是不变的。

纯函数式编程语言中的**变量**也不是命令式编程语言中的变量，即存储状态的单元，而是代数中的变量，即一个值的名称。变量的值是**不可变**的**（immutable）**，也就是说不允许像命令式编程语言中那样多次给一个变量赋值。比如说在命令式编程语言我们写“x = x + 1”，这依赖可变状态的事实，拿给程序员看说是对的，但拿给数学家看，却被认为这个等式为假。
函数式语言的如条件语句，循环语句也不是命令式编程语言中的**控制语句**，而是函数的语法糖，比如在Scala语言中，**if else**不是语句而是三元运算符，是有返回值的。
严格意义上的函数式编程意味着不使用可变的变量，赋值，循环和其他命令式控制结构进行编程。

函数式编程关心数据的映射，命令式编程关心解决问题的步骤，这也是为什么“函数式编程”叫做“函数式编程”。

二 实例

假如，现在你来到 baidu面试，面试官让你把number =[2, -5, 9, -7, 2, 5, 4, -1, 0, -3, 8]中的正数的平均值，你肯定可以写出：

```
#计算数组中正整数的平均值

number =[2, -5, 9, -7, 2, 5, 4, -1, 0, -3, 8]
count = 0
sum = 0

for i in range(len(number)):
    if number[i]>0:
        count += 1
        sum += number[i]

print sum,count

if count>0:
    average = sum/count

print average

```

```
#========输出===========
30 6
5
```

首先循环列表中的值，累计次数，并对大于0的数进行累加，最后求取平均值。

这就是命令式编程——你要做什么事情，你得把达到目的的步骤详细的描述出来，然后交给机器去运行。

这也正是命令式编程的理论模型——图灵机的特点。一条写满数据的纸带，一条根据纸带内容运动的机器，机器每动一步都需要纸带上写着如何达到。

那么，不用这种方式如何做到呢?

```
number =[2, -5, 9, -7, 2, 5, 4, -1, 0, -3, 8]

positive = filter(lambda x: x>0, number)

average = reduce(lambda x,y: x+y, positive)/len(positive)

print average

#========输出===========
5
```

这段代码最终达到的目的同样是求取正数平均值，但是它得到结果的方式和 之前有着本质的差别：通过描述一个列表->正数平均值 的映射，而不是描述“从列表得到正数平均值应该怎样做”来达到目的。

再比如，求阶乘

通过Reduce函数加lambda表达式式实现阶乘是如何简单:

```
from functools import reduce
print (reduce(lambda x,y: x*y, range(1,6)))
```

又比如，map()函数加上lambda表达式(匿名函数)可以实现更强大的功能:

```
squares = map(lambda x : x*x ,range(9))
print (squares)#  <map object at 0x10115f7f0>迭代器
print (list(squares))#[0, 1, 4, 9, 16, 25, 36, 49, 64]
```

三 函数式编程有什么好处呢?

1）代码简洁，易懂。
2）无副作用

由于命令式编程语言也可以通过类似函数指针的方式来实现高阶函数，函数式的最主要的好处主要是不可变性带来的。没有可变的状态，函数就是引用透明（Referential transparency）的和没有副作用（No Side Effect）。

Yuan先生
关注 - 1
粉丝 - 3925
我在关注他 取消关注

20 0

posted @ 2016-09-01 06:33 Yuan先生 阅读(11744) 评论(2) 编辑 收藏

Post Comment

#1楼 2016-09-05 23:55 | qinghean123 回复 引用

讲的很好~

支持(4) 反对(3)

#2楼 2018-01-21 20:32 | 西园公子zwj 回复 引用

讲的很好，很喜欢。

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

相关博文:
· Py徐少攻关之基础(3)
· Py西游攻关之多线程(threading模块)
· Py西游攻关之多进程(multiprocessing模块)
· 袁老师Py西游攻关之基础数据类型
· Py西游攻关之模块_Yuan先生新闻管理
» 更多推荐...

最新 IT 新闻:
· 征战全球16年：中国手机出海简史
· 都想做Netflix，但优爱腾需要自己的个性
· 蚂蚁上市，不应该只看它诞生了多少个千万富翁
· 任正非：通过3-5年把一批平庸或惰怠的干部更替掉
· 一个月均摊七八块钱，相互宝分摊金上涨背后到底发生了什么?
» 更多新闻...