一、函数基础

**1、函数的传参过程**

1）普通参数

def set\_age(name, age=20, com='hello'):

print('%s is %s years old. %s' % (name, age, com))

if \_\_name\_\_ == '\_\_main\_\_':

#以下写法都是正确的。

set\_age('bob', 23)

set\_age(age=23, name='bob')

set\_age('bob', com='hi')

2）参数组

def myfunc(\*args): # \*表示args是个元组

print(args)

def myfunc2(\*\*kwargs): # \*\*表示kwargs是个字典

print(kwargs)

if \_\_name\_\_ == '\_\_main\_\_':

myfunc('hello', 123)

# 参数'hello', 123将会传到元组中，args = ('hello', 123)

# 输出('hello', 123)

myfunc2(name='bob', age=23)

# 输出{'name': 'bob', 'age': 23}

3）序列对象

def add(x, y):

print(x + y)

if \_\_name\_\_ == '\_\_main\_\_':

nums = [10, 20]

add(\*nums)

**2、匿名函数**

**lambda** arg1, arg2... : expression

案例：简单的加减法数学游戏

cmds = {'+':lambda x, y: x + y, '-':lambda x, y: x - y}

nums = [random.randint(1,100) for i in range(2)]

nums.sort(reverse=True)

op = random.choice('+-')

result = cmds[op](\*nums)

#使用lambda匿名函数定义简单的函数

#x, y为参数，返回值x+y

**3、filter、map函数**

1）**filter**(func, seq)

# func必须是布尔函数（返回值是0/非零，或True/False）

# func历遍seq序列中的元素，返回True则保留该元素到filter对象

# filter返回类型为filter object

nums = [i for i in range(5)]

filterobj = filter(lambda x: x \* 2, nums)

print(list(filterobj))

#结果为：[1, 2, 3, 4]

2）**map**(func, seq)

# func历遍seq序列中的元素，保留返回结果到map对象

nums = [i for i in range(5)]

mapobj = map(lambda x: x \* 2, nums)

print(list(mapobj))

#结果为：[0, 2, 4, 6, 8]

二、函数高级应用

**1、变量作用域**

全局变量：函数外创建，存活至脚本运行结束，所有函数都可访问其值。

局部变量：作用域在函数内

全局与局部重名：局部变量会把全局变量遮盖，直至该函数结束。

函数内修改全局变量：在函数内使用global

名字空间：标识符的搜索顺序，局部 -> 全局 -> 内建

**2、函数式编程**

**1）偏函数**

作用：为函数参数赋默认值

from functools import partial

def add(a, b, c, d):

print(a + b + c + d)

if \_\_name\_\_ == '\_\_main\_\_':

myadd = partial(add, 10, 20, 30)

#或者myadd2 = partial(add, b=10, c=20, d=30)

myadd(5)

#其中，partial()中的add为自定义函数名

#若默认参数非顺序，可以按myadd2的方式添加，调用时myadd(b=5)

案例：简单GUI程序

import tkinter

**2）递归函数**

概念：函数内部包含了对自己的调用

案例：计算阶乘、简单排序算法、ls -R

**3）生成器**

概念：普通函数只能返回一个值，生成器能暂停执行函数并返回一个中间结果

def mygen():

yield 10

yield 'hello world'

if \_\_name\_\_ == '\_\_main\_\_':

a = mygen()

print(a.\_\_next\_\_()) #打印10

print(a.\_\_next\_\_()) #打印'hello world'

print(a.\_\_next\_\_()) #异常：StopIteration

for i in a:

print(i) #打印10和'hello world'，无异常

ips = ('176.121.202.%s' % i for i in range(1, 255))

#创建IP地址的生成器对象（与列表解析相似，但节省内存）

Python百例第68题：一次读取文件10行

**4）迭代器**

迭代器是一个Iterator对象，表示的是一个数据流，可以被next()函数调用并不断返回下一个数据，直到没有数据时抛出StopIteration错误。

可以把这个数据流看做是一个有序序列，但我们却不能提前知道序列的长度，只能不断通过next()函数实现按需计算下一个数据。因此迭代器甚至可以表示一个非常大的数据流，在for循环历遍时使用迭代器可以有效节省内存空间。(因为for循环本身就是不断调用next()函数)

>>>list=[1,2,3,4]

>>> it = iter(list) #创建迭代器对象

>>> it2 = (x for x in range(5)) #创建迭代器对象

>>> print (next(it)) #输出迭代器的下一个元素

>>> for x in it: #历遍迭代器

print (x, end=" ")

**3、内部函数**

**1）闭包**

def say\_hi(word):

def greet():

lb1.config(text = word)

return greet

#返回一个函数，而不是值

#以下为使用tkinter做窗体，不必掌握

root = tkinter.Tk()

lb1 = tkinter.Label(root, text='Hello World!', font='20')

myButton = partial(tkinter.Button, root, fg='white', bg='blue')

b1 = myButton(text = 'Button 1', command=say\_hi('zhuzhu'))

b2 = myButton(text = 'Button 2', command=say\_hi('honey'))

b3 = myButton(text = 'Button 3', command=root.quit)

#command是指定要执行的函数，而不是获取返回值

for item in (lb1, b1, b2, b3):

item.pack()

root.mainloop()

**2）装饰器**

def deco(func):

def color():

return '\033[5;31m%s\033[0m' % func()

return color

def hello():

return 'Hello world!'

@deco

#将函数welcome作为deco()参数

#相当于welcome = deco(welcome)，welcome() = color()

def welcome():

return 'Welcome!'

if \_\_name\_\_ == '\_\_main\_\_':

hello = deco(hello)

print(hello())

print(welcome())

注意：welcome为函数对象，welcome()为值，传参时需注意二者的区分。

**案例：测试程序运行效率**

1）有个程序包含多个函数

2）程序运行耗时较长

3）为了确定哪个函数是瓶颈,需要计算出每个函数运行时间

4）要求使用装饰器实现

import time

def deco(func):

def cnttime():

start = time.time()

func()

end = time.time()

print(end - start)

return cnttime

@deco

def myadd():

result = 0

for i in range(100000000):

result += i

print(result)

if \_\_name\_\_ == '\_\_main\_\_':

myadd()

**Python百例79题、装饰器任意颜色**