**17Python\_07函数基础+函数高级应用+模块**

**一 函数基础-创建函数**

**1.1 def语句**

• 函数用def语句创建,语法如下:

def function\_name(arguments):

"function\_documentation\_string"

function\_body\_suite

• 标题行由def关键字,函数的名字,以及参数的集合(如果有的话)组成

• def子句的剩余部分包括了一个虽然可选但是强烈推荐的文档字串,和必需的函数体

**1.2 前向引用**

• 函数不允许在函数未声明之前对其进行引用或者调用

def foo():

print('in foo')

bar()

foo() #调用函数报错,因为bar没有定义

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

def foo():

print('in foo')

bar()

def bar():

print('in bar’)

foo() #调用函数正常执行,虽然bar的定义在foo定义后面

**二 函数基础-调用函数**

**2.1 函数操作符**

• 使用一对圆括号()调用函数,如果没有圆括号,只是对函数的引用

• 任何输入的参数都必须放置在括号中

>>> def foo():

... print('Hello world!')

...

>>> foo()

Hello world!

>>> foo

<function foo at 0x7f18ce311b18> #引用返回函数在内存中的地址

**2.2 关键字参数**

写为key=val形式的参数称作关键字参数

直接写为arg形式的参数称作为位置参数

• 关键字参数的概念仅仅针对函数的调用

• 这种理念是让调用者通过函数调用中的参数名字来区分参数

• 这样规范允许参数缺失或者不按顺序

>>> def fun1(name,age):

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

>>> fun1('bob',20) # OK

bob is 20 years old.

>>> fun1(20,'bob') # 语法正确,语义不正确

20 is bob years old.

>>> fun1(age=20,name='bob') # OK,传参时不按顺序

bob is 20 years old.

>>> fun1(age=20,'bob') # 语法错误,关键字参数必须在后

File "<input>", line 1

SyntaxError: positional argument follows keyword argument

>>> fun1(20,name='bob') # Error,name得到了多个值

Traceback (most recent call last):

File "<input>", line 1, in <module>

TypeError: fun1() got multiple values for argument 'name'

>>> fun1('bob',age=20) # OK,传参时参数缺失

bob is 20 years old.

**2.3 参数组**

**定义函数时,参数名前面加上\*表示使用元组接收参数**

**定义函数时,参数名前面加上\*\*表示使用字典接收参数**

**传参时,使用\*表示把序列(字符串 列表 元组)对象拆开**

**传参时,使用\*\*表示把字典对象拆开**

• python允许程序员执行一个没有显式定义参数的函

• 相应的方法是通过一个把元组(非关键字参数)或字典(关键字参数)作为参数组传递给函数

func(\*tuple\_grp\_nonkw\_args, \*\*dict\_grp\_kw\_args)

>>> def fun1(\*args): #定义函数,使用元组args接收参数

... print(args)

...

>>> fun1() #调用函数,不传参,输出为空元组

()

>>> fun1('a') #调用函数,传1个参数,输出为1元素元组

('a',)

>>> fun1('a','b','c','1','2','3') #调用函数,传多个参数

('a', 'b', 'c', '1', '2', '3') #输出为多元素元组

>>> def fun2(\*\*kwargs): #定义函数,使用字典kwargs接收参数

print(kwargs)

>>> fun2()

{}

>>> fun2(name='bob')

{'name': 'bob'}

>>> fun2(name='bob',age=20)

{'name': 'bob', 'age': 20}

>>> def fun3(x,y):

... return x+y

...

>>> nums=[10,20]

>>> fun3(\*nums) #相当于fun3(10,20) 传递列表变量

30

>>> def fun4(name,age):

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

...

>>> adict={'name':'bob','age':20}

>>> fun4(\*\*adict) #传递字典变量

bob is 20 years old.

**案例:简单的加减法数学游戏**

1. 随机生成两个100以内的数字; 2. 随机选择加法或是减法

3. 总是使用大的数字减去小的数字; 4. 如果用户答错三次,程序给出正确答案

from random import randint, choice

def add(x, y):

return x + y

def sub(x, y):

return x - y

def exam():

cmds = {'+': add, '-': sub}

# 生成2个随机整数的列表

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

# 对列表进行降序排序

nums.sort(reverse=True)

# 生成操作符

op = choice('+-')

# 计算标准答案

result = cmds[op](\*nums)

# 生成算式,提供给使用者,并将使用者的答案赋值给answer

prompt = '%s %s %s =' % (nums[0], op, nums[1])

counter = 0

# 对answer进行对错判断

while counter < 3:

try:

answer = int(input(prompt))

except: # 可以补货所有异常,但是不推荐

print()

continue

if answer == result:

print('very good!')

break

print('you are wrong!')

counter += 1

else: # while也有else,在while没执行break语句时执行

print('正确答案是: %s%s' % (prompt, result))

def main():

while 1:

exam()

try:

yn = input('continue(y/n)?').strip()[0]

except IndexError: # 回车表示y

yn = 'y'

except (KeyboardInterrupt, EOFError):

yn = 'n'

if yn in 'nN':

print('Bye!')

break

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

main()

**三 函数基础-匿名函数**

**3.1 lambda**

• python允许用lambda关键字创造匿名函数

• 匿名是因为不需要以标准的def方式来声明

• 一个完整的lambda“语句”代表了一个表达式,这个表达式的定义体必须和声明放在同一行

函数名 = lambda [arg1[, arg2, ... argN]]: expression

>>> a = lambda x, y: x + y

>>> print(a(3, 4))

7

>>> def add(x,y): #正常函数

return x+y

>>> myadd = lambda x,y:x+y #匿名函数

>>> add(5,6)

11

>>> myadd(5,6)

11

**上例可不用定义add sub两个函数,将add sub两个函数的功能通过匿名函数的定义到cmds字典里**

from random import randint, choice

def exam():

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

# 生成2个随机整数的列表

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

# 对列表进行降序排序

nums.sort(reverse=True)

# 生成操作符

op = choice('+-')

# 计算标准答案

result = cmds[op](\*nums)

# 生成算式,提供给使用者,并将使用者的答案赋值给answer

prompt = '%s %s %s =' % (nums[0], op, nums[1])

counter = 0

# 对answer进行对错判断

while counter < 3:

try:

answer = int(input(prompt))

except: # 可以补货所有异常,但是不推荐

print()

continue

if answer == result:

print('very good!')

break

print('you are wrong!')

counter += 1

else: # while也有else,在while没执行break语句时执行

print('正确答案是: %s%s' % (prompt, result))

def main():

while 1:

exam()

try:

yn = input('continue(y/n)?').strip()[0]

except IndexError: # 回车表示y

yn = 'y'

except (KeyboardInterrupt, EOFError):

yn = 'n'

if yn in 'nN':

print('Bye!')

break

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

main()

**3.2 filter()函数**

• 接受2个参数;第1个参数是函数,如function;第2个参数是序列对象;

• function必须接收1个参数,返回值必须是True或False

• filter函数工作是,将序列对象中的每个值作为function的参数进行过滤,结果为真的保留,为假的舍弃

from random import randint

def fun1(x):

return x%2 # 结果只有1和0两种情况,1为真,0为假

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

nums=[randint(1,100) for i in range(10)]

print(nums)

result=filter(fun1,nums)

print(list(result))

result2=filter(lambda x:x%2,nums)

print(list(result2))

**3.3 map函数**

• 接受2个参数,map(func,seq)第1个参数是函数,;第2个参数是序列对象;

• func必须接收1个参数

map.py

import random

def func(x):

return x\*2

nums=[random.randint(1,100) for i in range(10)]

print(nums)

result1=map(func,nums)

print(list(result1))

result2=map(lambda x:x\*2,nums)

print(list(result2))

[26, 8, 59, 17, 84, 74, 41, 16, 11, 43]

[52, 16, 118, 34, 168, 148, 82, 32, 22, 86]

[52, 16, 118, 34, 168, 148, 82, 32, 22, 86]

**四 函数高级应用-变量作用域**

**4.1 全局变量**

**• 在函数外面定义的变量是全局变量.全局变量从定义开始,到程序结束,任意地方可见 可用.**

**• 在函数里面定义的变量是局部变量.局部变量只能在函数内部可见 可用.**

**• 如果全局变量和局部变量有同名,则局部变量将会遮盖主全局变量.**

• 标识符的作用域是定义为其声明在程序里的可应用范围,也就是变量的可见性

• 在一个模块中最高级别的变量有全局作用域

• 全局变量的一个特征是除非被删除掉,否则它们的存活到脚本运行结束,且对于所有的函数,他们的值都是可以被访问的

>>> x=10 #全局变量

>>> def func1():

... x = 'abc' #局部变量

... print(x)

>>> func1() #使用的是局部变量的值

abc

>>> x #在全局下调用变量x,使用的是全局变量的值

10

**4.2 局部变量**

• 局部变量只时暂时地存在,仅仅只依赖于定义它们的函数现阶段是否处于活动

• 当一个函数调用出现时,其局部变量就进入声明它们的作用域。在那一刻,一个新的局部变量名为那个对象创建了

• 一旦函数完成,框架被释放,变量将会离开作用域

• 如果局部与全局有相同名称的变量,那么函数运行时,局部变量的名称将会把全局变量名称遮盖住

>>> def func2():

... a=100

... print(a)

...

>>> func2()

100

**4.3 global语句**

• 因为全局变量的名字能被局部变量给遮盖掉,所以为了明确地引用一个已命名的全局变量,必须使用global语句

>>> x=10 #全局变量初始赋值

>>> def func3():

... global x #使用global调用全局变量

... x=10000 #给全局变量赋值

... print(x)

...

>>> func3()

10000

>>> x #全局变量的值在函数中被调用时已经被修改

10000

**五 函数高级应用-函数式编程**

**5.1 偏函数**

• 改造现有函数,生成新函数

• 改造时,将现有函数的一些参数固定

• 偏函数的概念: 将函数式编程的概念,和 固定值参数以及可变值参数 结合在一起

• 一个带有多个参数的函数,如果其中某些参数基本上固定的,那么就可以通过偏函数为这些参数赋默认值

>>> def func5(a,b,c,d,e): #定义函数必须有5个参数

... return a+b+c+d+e

...

>>> func5(10,20,30,40,50) #每次传参时前4个参数一样

150

>>> func5(10,20,30,40,60)

160

改造:

>>> from functools import partial

>>> myadd=partial(func5,10,20,30,40)

#定义新函数myadd(),在func5的基础上将前4项参数固定

>>> myadd(5)

105

>>> myadd(6)

106

>>> int('10',base=10)

10

>>> int('10',base=2)

2

>>> int('11001010',base=2)

202

>>> int2=partial(int,base=2) #改造int为偏函数int2

>>> int2('1010')

10

>>> int2('1111')

15

**5.2 递归函数**

• 如果函数包含了对其自身的调用,该函数就是递归的

• 在操作系统中,查看某一目录内所有文件、修改权限等都是递归的应用

**• 函数内部包含对自己的调用**

5\*func1(4)

4\*func1(3)

3\*func1(2)

2\*func1(1)

1

数字的阶乘

5!=5\*4\*3\*2\*1

4!=4\*3\*2\*1

5!=5\*4!=5\*4\*3!=5\*4\*3\*2!=5\*4\*3\*2!

1!=1

一个数字的阶乘,就是这个数乘以它下一个数的阶乘,1的阶乘是1.

>>> def func1(x):

... if x==1:

... return 1

... return x\*func1(x-1)

...

>>> func1(5)

120

**案例:快速排序**

1. 随机生成10个数字;2.利用递归,实现快速排序

from random import randint

def qsort(seq):

# 如果对象的长度是0或1,那么直接返回,不用排序

if len(seq) < 2:

return seq

# 假设第一个数值中间值,比它小的放到一个列表,比它大的放到另一个列表

middle = seq[0]

smaller = []

bigger = []

for i in seq[1:]:

if i <= middle:

smaller.append(i)

else:

bigger.append(i)

# 把各项从小到大拼接.对拼接项进行函数递归

return qsort(smaller) + [middle] + qsort(bigger)

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

nums = [randint(1, 100) for i in range(10)]

print(nums)

print(qsort(nums))

**5.3 生成器**

• **可以通过生成器表达式得到生成器**

>>> nums=(randint(1,100) for i in range(10))

>>> nums

<**generator object** <genexpr> at 0x7feb7e94a9e0>

#生成器只能取一次值

>>> for i in nums:

... print(i,end=’’)

...

8 18 96 75 97 6 74 876 72

**• 生成器通过yield语句,返回多个中间结果**

>>> def mygen():

... yield 10

... a=100+50

... yield a

... yield 200

...

>>> mg=mygen()

>>> list(mg)

[10, 150, 200]

**六 模块-模块和文件**

**6.1 什么是模块**

• 模块支持从逻辑上组织python代码

• 当代码量变得相当大的时候, 最好把代码分成一些有组织的代码段

• 代码片段相互间有一定的联系,可能是一个包含数据成员和方法的类,也可能是一组相关但彼此独立的操作函数

• 这些代码段是共享的,所以python允许“调入”一个模块,允许使用其他模块的属性来利用之前的工作成果,实现代码重用

**6.2 模块文件**

• 说模块是按照逻辑来组织python代码的方法,文件是物理层上组织模块的方法

• 一个文件被看作是一个独立模块,一个模块也可以被看作是一个文件

• 模块的文件名就是模块的名字加上扩展名.py

**七 模块-导入模块**

**7.1 搜索路径**

• 导入模块时,Python到指定路径下搜寻

• sys.path定义的路径

• PYTHONPATH环境变量定义的路径

>>> import sys

>>> sys.path

['/root/bin/pycharm2017/pycharm-2017.3/helpers/pydev', '/root/bin/pycharm2017/pycharm-2017.3/helpers/pydev', '/usr/local/lib/python38.zip', '/usr/local/lib/python3.8', '/usr/local/lib/python3.8/lib-dynload', '/root/nsd1906/lib/python3.8/site-packages', '/root/bin/pycharm2017/pycharm-2017.3/helpers/pycharm\_matplotlib\_backend', '/root/xyy/07']

(nsd1906) [root@room9pc01 07]# cp qsort.py /var/tmp/

(nsd1906) [root@room9pc01 07]# cd /tmp/

(nsd1906) [root@room9pc01 tmp]# python3

>>> import qstat #报错

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

(nsd1906) [root@room9pc01 tmp]# export PYTHONPATH=/var/tmp

(nsd1906) [root@room9pc01 tmp]# python3

>>> import qsort #导入成功

**7.2 模块导入方法**

• 使用import导入模块

• 可以在一行导入多个模块,但是可读性会下降

• 可以只导入模块的某些属性

• 导入模块时,可以为模块取别名

>>> import time, os, sys

>>> from random import choice

>>> import pickle as p

**7.3 导入和加载**

• 当导入模块时,模块的顶层代码会被执行

• 一个模块不管被导入(import)多少次,只会被加载(load)一次

**八 模块-内置模块**

**8.1 hashlib模块**

用于计算哈希值

>>> import hashlib

>>> m=hashlib.md5(b'12456')

>>> m.hexdigest()

'6a9edcb7b63821802aa44d35d531c9fc'

>>> with open('/etc/passwd','rb') as fobj:

... data=fobj.read()

...

>>> m=hashlib.md5(data)

>>> m.hexdigest()

'cc06d036add9c06649421e0cbcdfe8e2'

(nsd1906) [root@room9pc01 ~]# md5sum /etc/passwd

cc06d036add9c06649421e0cbcdfe8e2 /etc/passwd

>>> m=hashlib.md5()

>>> m.update(b'12')

>>> m.update(b'34')

>>> m.update(b'56')

>>> m.hexdigest()

'e10adc3949ba59abbe56e057f20f883e'

>>> m=hashlib.md5(b'123456')

>>> m.hexdigest()

'e10adc3949ba59abbe56e057f20f883e'

结果相同

**8.2 tarfile模块**

**用于压缩 解压缩**

**创建压缩文件**

>>> import tarfile #导入模块

>>> tar = tarfile.open('/tmp/mytest.tar.gz','w:gz')

#打开压缩包,不存在则创建

>>> tar.add('/etc/hosts') #添加文件到压缩包

>>> tar.add('/etc/passwd') #添加文件到压缩包

>>> tar.close() #关闭压缩包

(nsd1906) [root@room9pc01 ~]# tar -tvzf /tmp/mytest.tar.gz

-rw-r--r-- root/root 477 2019-01-24 13:03 etc/hosts

-rw-r--r-- root/root 2337 2019-11-07 09:14 etc/passwd

解压压缩包,打开时不用指定压缩格式

>>> tar=tarfile.open('/tmp/mytest.tar.gz') #打开压缩包

>>> tar.extractall(path='/var/tmp') #解压到指定位置,默认为当前目录

>>> tar.close()

**案例:备份程序**

1. 需要支持完全和增量备份; 2. 周一执行完全备份

3. 其他时间执行增量备份; 4. 备份文件需要打包为tar文件并使用gzip格式压缩

# (nsd1906) [root@room9pc01 day08]# mkdir /tmp/demo

# (nsd1906) [root@room9pc01 day08]# mkdir /tmp/demo/backup

# (nsd1906) [root@room9pc01 day08]# cp -r /etc/security /tmp/demo/

import os

import tarfile

import hashlib

import pickle

from time import strftime

def check\_md5(fname):

# 计算文件md5值的函数,接收文件名,返回md5值

m = hashlib.md5()

with open(fname, 'rb') as fobj:

while 1:

data = fobj.read(4096)

if not data:

break

m.update(data)

return m.hexdigest()

def full\_backup(src, dst, md5file):

# 拼接出备份的绝对路径

fname = os.path.basename(src)

fname = '%s\_full\_%s.tar.gz' % (fname, strftime('%Y%m%d'))

fname = os.path.join(dst, fname)

# 完全备份,就是把这个目录压缩

tar = tarfile.open(fname, 'w:gz')

tar.add(src)

tar.close()

# 计算每个文件的md5值

md5dict = {}

for path, folders, files in os.walk(src):

for file in files:

key = os.path.join(path, file)

md5dict[key] = check\_md5(key)

# 将md5值存入文件

with open(md5file, 'wb') as fobj:

pickle.dump(md5dict, fobj)

def incr\_backup(src, dst, md5file):

# 拼接出备份的绝对路径

fname = os.path.basename(src)

fname = '%s\_full\_%s.tar.gz' % (fname, strftime('%Y%m%d'))

fname = os.path.join(dst, fname)

# 计算每个文件的md5值

md5dict = {}

for path, folders, files in os.walk(src):

for file in files:

key = os.path.join(path, file)

md5dict[key] = check\_md5(key)

# 取出前一天的md5值

with open(md5file, 'rb') as fobj:

old\_md5 = pickle.load(fobj)

# 找出新增文件和改动文件,进行备份

tar = tarfile.open(fname, 'w:gz')

for key in md5dict:

if md5dict[key] != old\_md5.get[key]:

tar.add(key)

tar.close()

# 更新md5文件

with open(md5file, 'wb') as fobj:

pickle.dump(md5dict, fobj)

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

src = '/tmp/demo/security'

dst = '/tmp/demo/backup'

md5file = '/tmp/demo/backup/md5.data'

if strftime("%a") == 'Mon':

full\_backup(src, dst, md5file)

else:

incr\_backup(src, dst, md5file)