**17Python\_03文件对象+函数基础+模块基础**

**一 文件对象-文件打开方法**

**1.1 open内建函数**

• 作为打开文件之门的“钥匙”,内建函数open() 提供了初始化输入/输出(I/O)操作的通用接口

• 成功打开文件后时候会返回一个文件对象,否则引发一个错误

• 基本语法:

file\_object = open(file\_name, ‘mode’[, buffering=-1])

**1.2 文件对象访问模式**



**二 文件对象-文件输入[理解指针概念]**

**2.1 read方法**

**• read()方法用来直接读取字节到字符串中,最多读取给定数目个字节**

**• 如果没有给定size参数(默认值为-1)或者size值为负,文件将被读取直至末尾**

>>> data = fobj.read()

>>> print(data)

**2.2 readline方法[行结束符以’\n’表示]**

**读取打开文件的一行(读取下个行结束符之前的所有字节)**

**• 然后整行,包括行结束符,作为字符串返回**

**• 它也有一个可选的size参数,默认为-1,代表读至行结束符**

**• 如果提供了该参数,那么在超过size个字节后会返回不完整的行**

>>> data =fobj.readline()

>>> print(data)

**2.3 readlines方法**

**readlines()方法读取所有(剩余的)行然后把它们作为一个字符串列表返回**

>>> data = fobj.readlines()

>>> print(data)

**2.4 文件迭代**

**• 如果需要逐行处理文件,可以结合for循环迭代文件**

**• 迭代文件的方法与处理其他序列类型的数据类似**

>>> fobj = open('star.py')

>>> for eachLine in fobj:

... print(eachLine, end= '') # end = ‘’ 表示不换行输出

**示例:**

**open默认以读的方式打开,文件不存在则报错**

>>> f = open('/root/password')

>>> f = open('/root/original-ks.cfg')

**将全部内容读出来,赋值给data**

>>> data = f.read()

>>> print(date)

**继续向后读,因为全部内容已读完,所以剩下的是空**

>>> data = f.read()

>>> data

''

**关闭文件**

>>> f.close()

**重新打开**

>>> f = open('/root/password')

**读取1字节[1个字母或英文字符占1个byte]**

>>> f.read(1)

'r'

**从文件指针位置开始,继续读取3字节**

>>> f.read(3)

'oot'

**继续从文件指针开始,读取1行**

>>> f.readline()

':x:0:0:root:/root:/bin/bash\n'

**继续从文件指针开始,读取剩余的所有行,并作为一个字符串列表输出**

>>> f.readlines()

**读取文件迭代**

>>> f.close()

>>> f = open('/root/password')

>>> for i in f:

... print(i) #逐行输出

**读取非文本文件,读取时会报错**

>>> f = open('/tmp/ls') #不是文本文件

>>> f.read(10) #读取时报错

**读取非文本文件,要加上 b**

>>> f = open('/tmp/ls','rb') #r表示读,b表示byte

>>> f.read(10) #读取前10个字节

b'\x7fELF\x02\x01\x01\x00\x00\x00'

>>> f.close()

**三 文件对象-文件输出**

**3.1 write方法**

write()内建方法功能与read()和readline()相反。它把含有文本数据或二进制数据块的字符串写入到文件中去

• 写入文件时,不会自动添加行结束标志,需要程序员手工输入

>>> fobj.write('Hello World!\n') #返回写入的字节数

13

**3.2 writelines方法**

• 和readlines()一样,writelines()方法是针对列表的操作

• 它接受一个字符串列表作为参数,将它们写入文件

• 行结束符并不会被自动加入,所以如果需要的话,必须在调用writelines()前给每行结尾加上行结束符

>>> fobj.writelines(['Hello World!\n', 'python programing\n'])

**示例:**

**写入文本文件时,以w方式打开,会将文件清空**

**以w打开文件,会把文件清空**

>>> f =open('/root/password','w')

**向文件中写入数据,返回写入字节数**

>>> f.write('hello world!\n')

13 #写入数据不足4k时,暂时不把内存中的数据向硬盘中写入

**立即将内存中的数据向硬盘中写入**

>>> f.flush()

**向文本中写入行,注意换行符 \n**

>>> f.writelines(['2nd line.', 'new aaa\n', '3rd line.\n'])

>>> f.close()

**写入非文本文件**

**打开非文本文件,不存在则创建**

>>> f = open('/tmp/aaa', 'wb')

**直接向非文本文件写入字符,会报错**

>>> f.write('你好')

**将’你好\n’转换为2进制,1个汉字占3个byte,换行符或字母占1个byte**

>>> hi = '你好\n'

>>> hi.encode()

b'\xe4\xbd\xa0\xe5\xa5\xbd\n'

**再将转换为2进制的’你好\n’写入非文本文件**

>>> f.write(hi.encode()) #返回写入的字节数

7

**四 文件对象-操作文件**

**4.1 with字句**

**• with语句是用来简化代码的**

**• 在将打开文件的操作放在with语句中,代码块结束后,文件将自动关闭**

**使用with打开文件,代码块结束后,自动执行f.close()命令,再次读取会报错**

>>> with open('/root/password') as f:

... f.readline()

...

‘hello world!\n’ #读取代码块结束,后自动执行f.close()命令

>>> f.readline() #再次读取报错,因为文件已经关闭了

**4.2 文件内移动**

**• seek(offset[, whence]):在不关闭文件的情况下,移动文件指针到不同的位置**

**offset是相对于某个位置的偏移量**

**whence的值,0表示文件开头,1表示当前指针位置,2表示文件的结尾**

**• tell():返回当前文件指针的位置**

>>> f = open('/tmp/test')

>>> f.tell() #此时指针位置为0

0

>>> f.read(5) #读取5个字节

'hello'

>>> f.tell() #再次查询指针位置,此时指针位置为5

5

>>> f.seek(6,1) #将指针从当前位置移动6个字节

11

**案例1:模拟cp操作**

1 创建cp.py文件; 2 将/bin/ls“拷贝”到/root/目录下; 3 不要修改原始文件

f1 = open('/bin/ls', 'rb')

f2 = open('/root/ls', 'wb')

data = f1.read()

f2.write(data)

f1.close()

f2.close()

代码问题:

不建议硬编码,把文件路径用变量表示

变量名应该有意义

如果文件很大,将占用很大内存

src\_fname = '/bin/ls'

dst\_fname = '/tmp/list2'

src\_fobj = open(src\_fname, 'rb')

dst\_fobj = open(dst\_fname, 'wb')

while 1:

data = src\_fobj.read(4096)

if data:

dst\_fobj.write(data)

else:

break

src\_fobj.close()

dst\_fobj.close()

src\_fname = '/bin/ls'

dst\_fname = '/tmp/list2'

src\_fobj = open(src\_fname, 'rb')

dst\_fobj = open(dst\_fname, 'wb')

while 1:

data = src\_fobj.read(4096)

#读取4096byte数据,读1次指针移动1次,不占内存

print(src\_fobj.tell())

if data == b'': #为二进制的空

#或if len(data) == 0: #长度为0

#或if not data: #空串为假,取反为真

break

dst\_fobj.write(data) #写1次指针移动1次

print(dst\_fobj.tell())

src\_fobj.close()

dst\_fobj.close()

**五 函数基础-函数基本操作**

**5.1 函数基本概念**

• 函数是对程序逻辑进行结构化或过程化的一种编程方式

• 将整块代码巧妙地隔离成易于管理的小块

• 把重复代码放到函数中而不是进行大量的拷贝,这样既能节省空间,也有助于保持一致性

• 通常函数都是用于实现某一种功能

**5.2 创建函数**

• 函数是用def语句来创建的,语法如下:

def function\_name(arguments): #arguments表示传递的参数

"function\_documentation\_string" #文档字符串

function\_body\_suite #函数代码块

• 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

• 定义函数时不会执行函数内的代码

**5.3 调用函数**

**• 同大多数语言相同,python用一对圆括号调用函数: 函数名()**

**• 如果没有加圆括号,只是对函数的引用**

>>> def foo(): #定义函数

... print('hello')

...

>>> foo() #调用函数

hello

>>> foo #引用函数,返回函数在内存中的地址

<function foo at 0x7ff2328967d0>

**不能将函数赋值给变量,否则变量返回为None**

**5.4 函数的返回值**

• 多数情况下,函数并不直接输出数据,而是向调用者返回值

• 函数的返回值使用return关键字

• 没有return的话,函数默认返回None

>>> def foo():

... res = 3 + 4

>>> i = foo()

>>> print i

None

函数内定义的变量为局部变量,只在函数中才生效

>>> def add():

... a = 10 + 5

... **return** 'hello world'

... n = add()

>>> print(n)

hello world

>>> def add():

... a = 10 + 5

... return a #返回的是 a = 10 + 5,不是15

>>> n = add()

>>> n

15

**六 函数基础-函数参数**

**6.1 定义参数**

**形式参数**

– **函数定义时,紧跟在函数名后(圆括号内)的参数**被称为形式参数,简称**形参**。由于它不**是实际存在变量**,所以又称**虚拟变量(形式上占个位置)**

**• 实际参数**

– 在主调函数中**调用一个函数时,函数名后面括弧中的参数(可以是一个表达式)称为“实际参数”**,简称**实参**

>>> def add(x, y): #x,y为形参

... return x + y

...

>>> add(5, 5) #5,5为实参

10

>>> add(5, 5, 5) #参数多了,报错

**6.2 传递参数**

• 调用函数时,实参的个数需要与形参个数一致

• 实参将依次传递给形参

def copy(src\_fname, dst\_fname):

src\_fobj = open(src\_fname, 'rb’)

dst\_fobj = open(dst\_fname, 'wb’)

while 1:

data = src\_fobj.read(4096

if data == b'':

break

dst\_fobj.write(data)

src\_fobj.close()

dst\_fobj.close()

copy('/etc/passwd', '/tmp/passwd'

copy('/etc/group', '/tmp/group')

或者:

a = input('输入源文件名:')

b = input('输入目标文件名:')

def copy(src\_fname, dst\_fname):

src\_fobj = open(src\_fname, 'rb')

dst\_fobj = open(dst\_fname, 'wb')

while 1:

data = src\_fobj.read(4096)

if data == b'':

break

dst\_fobj.write(data)

src\_fobj.close()

dst\_fobj.close()

copy(a, b)

**6.3 位置参数**

• 与shell脚本类似,程序名以及参数都以位置参数的方式传递给python程序

• 使用**sys模块的argv列表**接收并存储位置参数: **sys.argv[1] 相当于$1**

import sys

def copy(src\_fname, dst\_fname):

src\_fobj = open(src\_fname, 'rb')

dst\_fobj = open(dst\_fname, 'wb')

while 1:

data = src\_fobj.read(4096)

if data == b'':

break

dst\_fobj.write(data)

src\_fobj.close()

dst\_fobj.close()

copy(**sys.argv[1], sys.argv[2]**) #相当于shell中的$1,$2

room9pc01 day03]# python3 cp4.py /etc/hosts /tmp/hosts

sys.argv[0] sys.argv[1] sys.argv[2]

import sys

def copy(src\_fname, dst\_fname):

src\_fobj = open(src\_fname, 'rb')

dst\_fobj = open(dst\_fname, 'wb')

while 1:

data = src\_fobj.read(4096)

if data == b'':

break

dst\_fobj.write(data)

src\_fobj.close()

dst\_fobj.close()

if len(sys.argv) < 3:

print('usage: %s src dst' % sys.argv[0])

exit(1) #程序遇到exit就会彻底结束,并返回值1

copy(sys.argv[1], sys.argv[2])

room9pc01 day03]# python3 cp4.py /etc/hosts #测试参数不够的情况

usage: cp4.py src dst #返回位置参数小于3时的提示信息

room9pc01 day03]# echo $? #查看代码中设定的程序结束返回值

1

**6.4 默认参数**

• 默认参数就是声明了默认值的参数

• 因为**给参数赋予了默认值,所以在函数调用时,不向该参数传入值也是允许的**

>>> def pstar(num): #定义函数,未给参数赋予默认值

... print('\*'\*num)

...

>>> pstar(15) #调用函数并传参,按传递的参数执行

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

>>> pstar() #调用函数不传参,报错

>>> def pstar(num = 10): #定义函数,并给参数赋予默认值

... print('\*'\*num)

...

>>> pstar() #调用函数不传参,但有默认参数,按默认参数执行

\*\*\*\*\*\*\*\*\*\*

>>> pstar(20) #调用函数并传参,按传递的参数执行

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**七 模块基础-定义模块**

**7.1 模块基本概念**

当代码量增加时,可以考虑把代码放到不同的文件里

每个文件以.py作为结尾的文件被称为模块文件

文件是代码的物理组织形式

模块是代码的逻辑组织形式.模块文件,将扩展名.py去掉就是模块名

**7.2 创建模块**

• 模块物理层面上组织模块的方法是文件,每一个以.py作为结尾的python文件都是一个模块

• 模块名称切记不要与系统中已存在的模块重名

• 模块文件名字去掉后面的扩展名(.py)即为模块名

**创建模块文件 start.py**

hi = 'hello world!' #模块start中的hi属性

def pstart(n= 30): #模块start中的pstart属性

print('\*'\*n)

**创建调用文件call\_start.py**

import start #导入模块start

print('this is call\_start')

start.pstart() #调用模块start中的pstart属性

print(start.hi) #调用模块start中的hi属性

**执行调用文件call\_start.py**

room9pc01 day03]# python3 call\_start.py

this is call\_start

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

hello world!

**八 模块基础-使用模块**

**8.1 导入模块(import)**

**• 使用import导入模块**

**• 模块属性通过“模块名.属性”的方法调用**

**• 如果仅需要模块中的某些属性,也可以单独导入**

**直接导入:**

>>> **import** time

>>> time.ctime()

'Sat Nov 2 15:45:54 2019'

**单独导入:**

>>> **from** random **import** randint, choice

#单独导入random中的randint和choice属性

>>> randint(1, 10)

>>> choice('abcdefg')

'a'

**导入模块的同时,为模块创建别名:**

>>> import getpass **as** gp

>>> p = gp.getpass()

**8.2 模块加载(load)**

• 一个模块只被加载一次,无论它被导入多少次

只加载一次可以阻止多重导入时代码被多次执行

如果两个文件相互导入,防止了无限的相互加载

• 模块加载时,顶层代码会自动执行,所以只将函数放入模块的顶层是良好的编程习惯

**8.3 模块导入的特性**

**模块特性:** 导入模块时,模块内的代码将会先执行一遍,所以,若模块中有执行语句,在模块被导入时,模块内的执行语句会被先执行

使用\_\_name\_\_属性,实现 模块中有执行语句,希望运行模块时,执行模块中的执行语句;当作为模块被导入时,不执行模块中的执行语句.**[双下划线]**

**• 模块具有一个\_\_name\_\_特殊属性**

**• 当模块文件直接执行时,\_\_name\_\_的值为'\_\_main\_\_'**

**• 当模块被另一个文件导入时,\_\_name\_\_的值就是该模块的名字**

模块文件foo.py

print(\_\_name\_\_)

room9pc01 day03]# python3 foo.py

\_\_main\_\_ #模块文件直接执行时,\_\_name\_\_的值为\_\_main\_\_

调用文件bar.py

import foo

day03]# python3 bar.py

foo #当模块被调用,并在调用文件中被执行时,\_\_name\_\_的值为模块名

修改模块文件start.py

hi = 'hello world!'

def pstart(n= 30):

print('\*'\*n)

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

print(hi)

pstart(30)

直接执行start.py

hello world!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

调用文件call\_start.py

import start

print('this is call\_start')

start.pstart()

print(start.hi)

直接执行call\_start.py

this is call\_start

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

hello world!

因为在调用文件中start.py中,使用了if \_\_name\_\_ == '\_\_main\_\_':判断语句,在调用start模块时,\_\_name\_\_值为模块名start,不为\_\_main\_\_,不执行调用文件start.py中的print(hi)及pstart(30)语句.从而实现了: 运行模块时,执行模块中的执行语句;当作为模块被导入时,不执行模块中的执行语句.

if \_\_name\_\_ == '\_\_main\_\_': 快捷输入方法: main + tab

**案例: 生成随机密码**

• 创建randpass.py脚本,要求如下:

1. 编写一个能生成8位随机密码的程序

2. 使用random的choice函数随机取出字符

3. 改进程序,用户可以自己决定生成多少位的密码

from random import choice

str = '~!@#$%^&\*()\_+-=abcdefhijklmnopqstruvwxyz \

ABCEFGHIJKLMNOPQRSTUVWXYZ0123456789'

n = int(input('请输入密码位数:'))

password = ''

for i in range(n):

password\_str = choice(str)

password += password\_str

print(password)

**转成函数:**

from random import choice

from string import ascii\_letters,digits

all\_chs = ascii\_letters + digits

def randpass(n=8):

result = ''

for i in range(n):

ch = choice(all\_chs)

result += ch

return result

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

a = randpass()

print(a)