**python运行方式**

交互式解释器

进入交互式解释器

python3

退出交互式解释器

exit()货ctrl+d

文件形式

明确指定解释器

python3 hello.py

赋予python文件可执行权限

chmod +x hello.py

./hello.py

**python语法结构**

语句块缩进

python代码块通过缩进对齐表达式代码缩进而不是用大括号，缩进表达一个语句属于哪个代码块

缩进风格

1或2：可能不够，很难确定代码语句属于哪个代码块

8至10个：可能太多，如果代码内嵌的层次太多，就会使得代码很难读

4哥空格：非常流行。

缩进相同的一组语句构成一个代码块，称为代码组

首行以关键字开始，以冒号：结束，该行至后的一行或多行代码构成代码组，如果代码组只有一行，可以将其直接写在冒号后面，但是这样的可读性比较差，不推荐

注释和续行

注释以#开头，注释科一在一行的任何地方开始，解释器会忽略掉该行#至后的所有内容

一行过长的语句可以用反斜杠\分解成几行

同行多个语句

分号;允许将多个语句写在同一行上，但是这些语句不能在这行开始一个新的代码块，因为可读性会变差，所以不推荐使用

**python语法**

获取帮助

help(print)

输出语句print

print('hello world!') # 字符串必须放在引导中

print('hello', 'world', '!', 123) # 数字不用放在引号中

print('hello', 'world', '!', 123, sep='\*\*\*') # 各项间用\*\*\*分隔

print('hello' + 'world') # 字符串拼接

print('hello', 'world', '!', 123, end='') # 不打印回车

输入语句input

a = input('number:') # input读入的数据是字符类型

# a+5错误，不能把数字和字符相加

print(int(a) + 5) # 将字符转换成数字再加5

print(a + str(5)) # 将数字5转换成字符，再和字符串拼接

**python变量**

变量定义

变量名称约定

第一个字符只能是大小写字母或下划线

后续字符职能是大小写字母货数字或下划线

区分大小写

python是动态类型语言，即不需要预先声明变量的类型

推荐才用全名的办法

变量名全部采用小写字母

简短、有意义

多个单词间用下划线分割

变量名用名词，函数名用谓语（动词+名词）

类名采用驼峰形式

变量赋值

变量的类型喝值在赋值的那一刻被初始化

变量赋值通过等号来执行

python也支持增量赋值

a = 10 + 5 # 变量赋值，自右向左进行

a = a + 1 # 可以简化位以下形式

a += 10

# b+=10错误，因为等价于b=b+10,b没有赋值

运算符

标准算术运算符：+-\*/ // % \*\*

比较运算符：< <= > >= == !=

逻辑运算符：and not or

print(5 / 2) # 2.5

print(5 // 2) # 2

print(5 % 2) # 之遥余数，模运算，1

print(2 \*\* 3) # 2的3次方，8

20 > 10 > 5 # true相当于20>10and10>5

**python数据类型**

数字

基本数字类型

int：有符号整数

bool：布尔值（True：1，False：0）

float：浮点数

数字表示方法

python默认使用十进制数表示；数字以0o或者0O开头表示为8进制数；数字以0x或0X开头表示为16进制数，数字以0b或0B开头表示为2进制数

a=True+2

print(a)#bool类型也是整型

print(0o11)#8进制数11换算成10进制数为9

print(0x11)#16进制数11换算成10进制数是17

print(0b11)#二进制数11换算成10进制数是3

字符串

定义字符串

python种字符串被定义为引号之间的字符集合，可以使用成对的单引号货双引号，两个意义相同，还支持三引号（三个连续的引号或者双引号），可以用来包含特殊字符。不区分字符和字符串

py=’python’

字符串切片（类似与截取）

使用索引运算符[]和切片运算符[:]可得到字符串，第一个字符串的索引是0,最后一个字符的索引是-1，字符串包含切片中的起始下标，但不包含下标，切片没有下标不会报错

py[0]#取出第一个字符

py[-1]#取出最后一个字符

py[2:4]#切片，取出2到4个字符

py[2:]#结尾不写，表示取到最后部分结束

py[:]#取所有

py[::-1]#步长为负，表示从右往左取

字符串连接

使用+号可以将多个字符串拼接在一起，使用\*号可以将一个字符串重复多次

py+’is good’

‘\*’ \* 50#打印\*50次

列表

可以将列表当成普通的“数组”，它能保存任意数量任意类型的python对象，像字符串一样，列表也支持下标和切片操作，列表中的项目可以改变

列表操作

使用in或not in判断成员关系，使用append方法向列表中追加元素

alist=[‘bob’,’alice’,10,[1,2,3]]

[‘tom’]+alist

alist.append(100)#向列表中追加元素

元组

可以认为元组是“静态”的列表，元组一旦定义，不能改变

aaa=(‘bob’,10,28)

aaa[2:]

(100,)+aaa

字典

字典是由键值对构成的映射数据类型，通过键取值，不支持下标操作

adict={‘name’:’zhangsan’,’age’:25}

25 in adict#false

adict[‘name’]#’zhangsan’

数据类型比较

按存储模型分类

标量类型：数值、字符串

容器类型：列表、元组、字典

按更新模型分类

可变类型：列表、字典

不可变类型：数字、字符串、元组

按访问模型分类

直接访问：数字

顺序访问：字符串、列表、元组

映射访问：字典

**python判断语句**

if语句

语法：if expression:

if\_suite

else:

else\_suite

如果表达式的值非0或者为布尔值true，则代码组if suite被执行；否则就去执行ekse suite，只要表达式数字为非零值即为true，空字符串、空列表、空元组、空字典的值均为false

if “”:

print(‘yes’)

else:

print(‘no’)

扩展if语句

语法：if expression1:

if\_suite

elif expression2:

elif\_suite

else:

else\_suite

条件表达式

语法：x if c else y中间是条件，两边是结果

a=10

b=5

s=a if a<b else b

print(s)

**python循环**

python中有while和for循环，循环次数未知的情况下，建议采用while循环，循环次数可以预知的情况下，可以采用for循环

while循环语句结构

while expression:

while\_suite

语句while\_suite会被不断的执行循环，直到表达式的值为-或FLASE

sum=0

counter=1

while counter<=100:

sum+=counter

counter+=1

print(sum)

while进阶

break语句

break跳出循环体，执行下面的语句

continue语句

当遇到continue语句时，程序循环终止当前循环，并忽略剩余的语句，然后回到循环的顶端，如果条件满足，循环体内的语句继续执行，如果不满足则退出

else语句

python中的while语句也支持else子句，else子句只在循环正常完成后执行，break语句会跳过else块

for循环详解

python中的for接受可迭代对象（例如迭代器或序列）作为其参数，每次迭代其中一个参数

for iter\_var in iterable:

suite\_to\_repeat

与while循环一样，支持break、continue、else语句

alist=[‘bob’,’alice’]

for name in alist:

print(name)

range函数

for循环常与range函数一起使用

range函数提供循环语法

range函数的完整语法为：

range(start,end,step=1)

list(range(10))#只写10，表示结束数字10，但是不包含10，开始数字是0

list(range(6,10))#[6,7,8,9,10]

list(range(1,10,2))#[1,3,5,7,9]

list(range(10,0,-1))#[10,9,8,7,6,5,4,3,2,1]

for i in range(1,101):

print(‘hello’)#输出三次hello

列表解析

是一个非常有用、简单、而且灵活的工具，科一动态创建列表

语法：

[expr for i in iterable]

[3+2 for i in range(10)]#循环控制3+2运行多少次,[5,5,5,5,5,5,5,5,5,5]

**文件对象**

文件打开方法

open及file内建函数

内建函数open()以及file()提供了初始化输入/输出（I/O）操作的通用接口

成功打开文件时候会返回一个对象，否则引发一个错误

open()方法和file()方法可以完全相互替换

基本语法：

aaa=open(file\_name,’r’,buffering=-1)

文件对象访问方法

r以读的方式打开（文件不存在则报错）

w以写方式打开（文件存在则清空，不存在则创建）

a以追加模式打开（必要时创建文件）

r+以读写模式打开（参见r）

w+以读写模式打开（参见w）

a+以读写模式打开（参见a）

b以二进制模式打开

文件输入

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，文件将读取至末尾

data=fobj.read()

print(data)

readline方法

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，有一个可选的size参数，默认为-1，代表读至行尾结束符，如果提供了该参数，那么在超过size个字符后返回不完整的行

data=fobj.readline()

print(data)

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)

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(‘str.py’)

for eachline in fobj:

print(eachline,end=’’)

forj=open(‘/tmp/mima’)

fobj.read(4)#读取四个字节，建议一次是1024的倍数

fobj.readline()#适合文本，读取一行

fobj.readlines()#适合文本，把所有行读到列表中

fobj.close()

文件输出

write方法

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

writelines方法

和readlines()一样，主要是针对列表的操作，它接受一个字符串列表作为参数，将它们写入文件，行结束符不会自动被加入，如果需要的话，必须在调用前每行结尾加上行结束符

fobj=open(‘/tmp/mima’,’w’)

fobj.write(‘hello world!\n’)#写数据时，先放到缓冲区

#当数据量达到一定程度时，自动写入磁盘

fobj.flush()#立即将数据同步到磁盘

fobj.writelines([‘3rd line.\n’,’new line.\n’])

fobj.close()#关闭文件，数据自动写入磁盘

操作文件

with子句

with语句是用来简化代码的，再将打开文件的操作放在with语句中，代码结束后，文件将自动关闭

with open(‘/tmp/mima’) as fobj:#with语句结束后，文件自动关闭

print(fobj.readline())

文件内移动

seek(offset[,whence]):移动文件指针到不同位置

offset是相当于某个位置的偏移量

whence的值，0表示文件开头，1表示当前位置，2表示文件的末尾

tell():返回当前文件指针的位置

fobj=open(‘/tmp/mima’,’rb’)#非文本要加上b，文本文件也可以加上b，表示byte

fobj.tell()#返回当前文件指针的位置

fobj.seek(10,0)#第一个数字是偏移量;第二个数字是相对位置，0表示开头，1是当前

fobj.read(4)#读到第四个字节，因为是bytes方式打开，所以显示b’xxxx'

fobj.seek(-5,2)#移动到文件倒数第5个字节的位置。以’rb’擦爱能写负数，’r’不行

标准文件

程序一执行，就可以访问三个文件

标准输入：一般是键盘，使用sys.stdin

标准输出：一般是显示器缓存输出，使用sys.stdout

标准错误：一般是显示器的非缓存输出，使用sys.stderr

import sys

a=sys.stdin.readline()#读键盘输入，回车也作为一个字符\n输入

print(a)

sys.stdout.write(a)

sys.stderr.write(a)

**函数基础**

创建函数

函数是用def语句来创建的，语法如下

def function\_name(arguments):

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符串，和必须的函数体

调用函数

同大多数语言相同，python用一对圆括号调用函数，如果没有加圆括号，只是对函数的引用

def foo():

print(‘hello’)

foo()

函数的返回值

形式参数：函数定义时，紧跟在函数名后（圆括号内）的参数被称为形参

实际参数：在主调函数中调用一个函数时，函数后面的圆括号中的参数（可以是一个表达式）称为实参

调用函数时，实参的个数需要与形参个数一致，实参将以此传递给形参

def foo(a,b):

print(‘a=%d,b=%d’ % (a,b))

foo(3,4)

位置参数

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

使用sys模块的argv列表接收

import sys

print sys.argv

默认参数

默认参数就是为了声明默认值的参数，因为给参数赋予了默认值，所以在函数调用时，不向该参数传值也是允许的

def pstar(num=10):

print(‘\*’ \* num)

pstar()

**模块基础**

创建模块

模块在物理层面上组织模块的方法是文件，每一个以.py结尾的python文件都是一个模块；模块名称不要与系统中已存在的模块重名；文件名字去掉后面的扩展名（.py）就是模块名

导入模块（import）

使用import导入，模块被导入后，程序会自动生成pyc的字节码文件以提高性能，模块通过“模块名.属性”的方法调用，如果仅需模块中的某些属性，也可以单独导入

import sys

import os,string

from random import randint

模块加载（load）

一个模块只允许被加载一次，无论被导入多少次；只加载一次可以阻止多重导入时代码被多次执行；模块夹杂爱时，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

模块的导入特性

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

vim foo.py

hi=’hello world’

if \_\_name\_\_ == ‘\_\_main\_\_’:

print(hi)

vim bar.py

import foo

**shell相关模块**

shutil模块

复制和移动

shutil.copyfileobj(fsrc,fdst[,length])

将类似文件的对象fsrc的内容复制到类似文件的对象fdst

shutil.copyfile(fsrc,fdst,\*,follow\_symlinks=True)

将文件src的文件内容（无元数据）复制到名为dst的文件，然后的返回dst

shutil.copy(src,dst,\*,follow\_symlinks=True)

将文件src复制到文件或目录dst。src和dst应为字符串

shutil.copy2(src,dst,\*,follow\_symlinks=True)

与copy相同，但copy2也尝试保留所有文件元数据

shutil.move(src,dst,copy\_function=copy2)

递归的将文件或目录移动到另一个位置，并返回目标

import shutil

# https://yiyibooks.cn/

f1 = open('/etc/passwd', 'rb')

f2 = open('/tmp/abcd', 'wb')

shutil.copyfileobj(f1, f2)

f1.close()

f2.close() # 复制出来/tmp/abcd

shutil.copyfile('/etc/passwd', '/tmp/xyz') # 复制出来xyz

shutil.copy('/etc/passwd', '/tmp') # 复制到tmp目录底下

shutil.copy('/etc/passwd', '/tmp/aabb') # 复制出来aabb

shutil.copy2('/etc/passwd', '/tmp/aabbcc') # 复制出来aabbcc，并且不改变权限

shutil.move('/tmp/abcd', '/home') # 移动abcd到另外一个地方

目录操作

shutil.copytree(src,dst,symlins=False,ignore=None,copy\_function=copy2,ignore\_dangling\_symlinks=False)

递归地复制以src为根的整个目录树，返回目标目录。目标目录不能已经存在

shutil.rmtree(path,ignore\_errors=Flase,onerror=None)

删除整个目录树；路径必须指向目录，而不是指向目录的符号链接

shutil.copytree('/home', '/asd') # 复制目录，目标目录必须不存在

shutil.rmtree('/asd') # 删除目录

权限管理

shutil.copymode(src,dst,\*,follow\_symlinks=True)

将权限位从src复制到dst。文件内容，所有者和组不受影响。src和dst是以字符串的形式给出路径名称

shutil.copystat(src,dst,\*,follow)symlinks=True)

将权限位，最后访问时间，上次修改时间和标志从src移动到dst

shutil.chown(path,user=None,group=None)

更改给定路径的所有用户和/或组

shutil.copymode('/home', '/asd') # 复制权限，目标必粗存在

shutil.copystat('/home', '/asd') # 复制访问时间等

shutil.chown('/asd', user='www', group='www') # 更改目录所属组所属主

**语法风格**

变量赋值（支持链式多重赋值）：x=y=10，a,b=10,20

合法标识符

第一个字符必须是字母或下划线，剩下的字符可以是数字、字母或下划线，区分大小写

关键字

关键字列表和iskeyword()函数都放入了keyword模块以便查询

内建

除了关键字外，python还可以在任何一级代码使用“内建”的名字集合，但是应该把它当做“系统保留字”，常见的系统保留字由：True、False、None

模块结构及布局

编写程序时，应该建立一种同一切容易阅读的结构，并将它应用到每一个文件中趣

起始行（使用什么翻译器#!/usr/bin/env python）

模块文档字符串

导入模块

全局变量声明

类定义

函数定义

程序主体（if \_\_name\_\_==’\_\_main\_\_’）

**字符串详解**

序列

序列操作符

seq[ind]获取下标为ind的元素

seq[ind1:ind2]获取下标从ind1到ind2的元素集合

seq \* expr序列重复expr次

seq1+seq2连接序列seq1和seq2

obj in seq判断obj元素是否包含在seq中

obj not in seq判断obj元素是否不包含在seq中

内建函数

list(iter)把可迭代对象转换为列表

str(obj)把obj对象转换成字符串

tuple(iter)把一个可迭代对象转换成一个元组

len(seq)返回seq的长度

max(iter,key=none)返回iter中的最大值

enumerate接收一个可迭代对象座位参数，返回一个enumerate对象

reversed(seq)：接收一个序列作为参数，返回一个有序的序列

sorted(iter)接受一个可迭代对象作为参数，返回一个有序的列表

str(100)

str([10, 20, 30])

max((10, 200, 30))

min((10, 200, 30))

users = ['bob', 'alice', 'join']

for i in range(len(users)):

print('#%s:%s' % (i, users[i]))

list(enumerate(users)) # 应用时不必转换

for item in enumerate(users):

print('#%s:%s' % (item[0], item[1]))

for ind, user in enumerate(users): # (0,bob),(1,alice)

print('#%s:%s' % (ind, users))

for u in reversed(users): # 翻转

print(u)

print(sorted(users)) # 排序

字符串

字符串操作符

比较操作符：字符串大笑按ASCII码值进行大小比较

切片字符串：[]、[:]、[::]

成员关系操作符：in、not in

格式化操作符

%c转换成字符

%s优先用str()函数进行字符串转换

%d%i转换成有符号十进制数

%o转换成无符号八进制数

%e/%E转换成科学计数法

%f/%F转换成浮点数

\*定义宽度或者小数点精度

-左对齐

+在正数前面显示+号

<sp>在正数前面显示空格

#在八进制数前面显示零0，在十六进制前面显示0x或者0X

0显示的数字前面填充0而不是默认的空格

'%s is %s years old' % ('bob', 25)

'%s is %d years old' % ('bpb', 25)

'%12s%8s' % ('name', 'age') # 总宽度是12个字符，name占了4个，还剩8个空格，默认右对齐

'%-12s%-8s' % ('name', 'age') # 总宽度是12个字符，name占了4个，还剩8个空格，默认左对齐

'%c' % 98 # ascii为8的字符

'%#o' % 30 # 8进制数形式

'%#x' % 30 # 16进制数形式

'%f' % (5 / 3) # 浮点数

'%5.2f' % (5 / 3) # 总宽度5位，小数占2位

'%+d' % 10 # 正数前面加+号

'% d' % 10 # 正数前面加空格

'%010d' % 55 # 总宽度为10，不够宽度不够，前面补0

format函数

使用位置参数

‘my name is {},age {}’.format(‘hoho’,18)

使用关键字参数

‘my name is {name},age is {age}’.format({‘name’:’bob’,’age’:23})

填充与格式化

{:[填充字符][对齐方式 <^>][宽度]}

使用索引

'name is {0[0]} age is {0[1]}'.format(['bob', 23])

'{} is {} years old'.format('bob',25)#'bob is 25 years old'

'{1} is {0} years old'.format(25,'bob')#'bob is 25 years old'

'{:<20}'.format('hello') # 'hello '

'{:>20}'.format('hello') # ' hello'

'{:^20}'.format('hello') # ' hello '

原始字符串操作符

原始字符串操作符是为了对付那些在字符串中出现的特殊字符，在原始字符串里，所有的字符都是直接按照字面的意思来使用，没有转义或不能打印字符

内建函数

string.capitalize():把字符串的第一个字符大写

string.center(width):返回一个源字符串居中，并使用空格填充至长度width的新字符串

string.count(str,beg=0,end=len(string)):返回str在string里面出现的次数，如果beg活着end指定则返回指定范围内str出现的次数

string.endswith(obj, beg=0,end=len(string)):检查字符串是否以obj结束，如果beg活着end指定则检查指定范围内是否以obj结束，如果是，则返回true，否则返回false

string.islower():如果string中包含至少一个区分大小写的字符，并且所有这些字符都是小写，则返回true，否则返回false

string.strip():删除string字符串两端的空白

string.upper():转换string中的小写字母为大写

string.split(str=””,num=string.count(str)):以str为分隔符切片string，如果num有指定，则仅分隔num个子字串

astr='\thello '

astr.split()#去除两端空白字符

astr.lstrip()#去除左端空白字符

astr.rsplit()#去除右端空白字符

'hello.tar.gz'.split('.')#['hello', 'tar', 'gz']，用点来切割，默认是空格，把切出来的放到列表里面

hi='hello world'

hi.title()#'Hello World'，每个单词字母大写，本身不会变，只会返回一个改完的

hi.upper()#'HELLO WORLD'，转成大写字母

hi.lower()#'hello world'，转成小写字母

hi.islower()#true，所有字母是不是小写，不关数字

'hao123'.isdigit()#false，所以都是数字，有不是数字的返回false

hi.isidentifier()#false，是不是标识符

hi.center(50)#' hello world '，居中，宽度是50

hi.center(50,'#')#'###################hello world####################'，用井号填充

hi.ljust(50)#'hello world '，实现左对齐

hi.rjust(50)#' hello world'，实现右对齐

hi.startswith('he')#true，是不是以he开头

hi.endswith('d')#true，是不是以d结尾，可以是多个字符

**列表和元组**

列表

创建及访问列表

列表是有序、可变的数据类型，列表中可以包含不同类型的对象，列表可以由[]或工厂函数创建，支持下标及切片操作

更新列表

通过下标只能更新值，不能使用标添加新值

列表内建函数

list.append(obj)向列表中添加一个对象obj

list.count(obj)返回一个对象obj在列表中出现的次数

list.extend(seq)把序列seq的内容添加到列表中

list.index(obj)返回obj对象的下标

list.insert(index.obj)在索引为index的位置插入对象obj

list.reverse()原地反转列表

list.sort()排序

alist=[1,50,60,100]

alist[0]=10#把第一项修改为10

alist[1:3]=[30,40]#修改第二项和第三项的值

alist[3:3]=[70,80,90]#在第三项后添加三个

alist.append(30)#在列表最后追加30

alist.sort()#排序，默认升序

alist.reverse()#排序，以降序排列

alist.count(30)#统计30出现了几回

alist.remove(30)#将第一个出现的30删除

alist.pop(2)#弹出下标为2的，默认弹出最后一个

alist.index(50)#返回50的下标

alist.extend('abc')#以列表的形式扩充进去

alist.extend(['abc','xyz'])

alist.insert(3,1000)#在下标为3的地方插入1000

元组

创建元组

通过()或工厂函数tuple()创建元组、元组是有序的、不可变类型、与列表类似，作用于列表的操作，绝大多数也可以作用域元组

单元素元组

如果一个元组只有一个元素，那么创建该元组的时候，需要加上一个逗号atuple=(‘hello’,)

更新元组

虽然元组本身是不可变的，但是因为它属于容器类型，也就意味着元组的某一个元素是可变的容器类型，那么这个元素中的项目仍然是可变的

atuple = ('bob', ['bob', 23]) # ('bob', ['bob', 23])

atuple[-1] = ['boy', 22] # 'tuple' object does not support item assignment

atuple[-1][-1] = 22 # ('bob', ['bob', 22])

atuple[-1].append(175) # ('bob', ['bob', 22, 175])

**字典和集合**

字典基础操作

创建字典

通过{}操作符创建字典

通过dict()工厂方法创建字典

通过fromkeys()创建具有相同值的默认字典

adict=dict([‘aa’,’cd’,(‘name’,’zhangsan’)])

bdict={}.fromkeys([‘bob’,’bob’,’alice’],7)

访问字典

字典是映射类型，意味着没有下标，访问字典中的值需要使用相应的键

for key in bdict:

print(‘%s:%s’ % (key,bdict[key]))

print(‘%(bob)s’ % bdict)#7

更新字典

通过键更新字典

如果字典中有该值，则更新相关值

如果字典中没有该值，则向字典中添加该值

bdict[‘tom’]=8

bdict[‘join’]=7

删除字典

通过del可以删除字典中的元素或整个字典

使用内部方法clear()可以清空字典

使用pop()方法可以弹出字典中的元素

bdict.pop(‘alice’)#把alice弹出，列表默认弹出最后一项

字典操作符

使用字典键查找操作符[]，查找所对应的值

使用in和not in判断键是否存在于字典中

7 in bdict#false

bob in bdict#true

字典相关函数

作用于字典的函数

len():返回字典中元素的数目

hash()：本身不是为字典设计，但是可以判断某个对象是否可以作为字典的键

len(bdict)

hash(3)

hash((‘a’,’b’))

hash([‘a’,’b’])

hash({‘a’,’b’})#列表和字典不可以哈希、元组、数字、字符串不可变的可以

字典的内建方法

dict.copy():返回字典的一个副本

dict.get(key,default=None):对字典中的key，返回它对应的值，如果不存在此键，则返回默认的值

dict.setdefault(key,default=None):如果字典中不存在key值，由dict[key]=default为它赋值

dict.items():返回一个包含字典中（键，值）对元组的列表

dict.keys()：返回一个包含字典种键的列表

dict.values():返回一个包含字典中所有值的列表

dict.update(dict2)：将字典dict2的键-值队添加到字典dict

cdict=bdict.copy()#将bdict的内容赋值给cdict，cdict使用全新的内存空间

bdict.get('bob')#返回bob对应的value，如果没有bob，默认返回none

bdict.get('jane','not found')#如果没有jabne，返回notfound

bdict.setdefault('bob',10)#bob已经是字典的key，返回value

bdict.setdefault('bobo',10)#bobo不是字典的key，向字典中写入

list(bdict.keys())#['bob', 'aaa', 'alice']

list(bdict.values())#[7, 7, 7]

list(bdict.items())#[('bob', 7), ('aaa', 7), ('alice', 7)]

bdict.update({'aaa':111,'bbb':222})#合并字典，{'bob': 7, 'aaa': 111, 'alice': 7, 'bbb': 222}

集合操作

创建集合（无值的字典）

数学上，把set称作由不同的 元素组成的集合，集合（set）的成员通常被称作集合元素

集合对象是一组无序排列的可哈希的值

集合有两种类型

可变集合set

不可变集合frozenset

集合类型操作符

支持用in和not in操作符检查成员

能够通过len()检查集合大小

能够使用for迭代集合成员

不能去切片，没有键

|：取并集

&：交集

-：差补

集合内建方法

set.add()：添加成员

set.update()：批量添加成员

set.remove()：删除成员

s.issubset(t)：如果s是t的自己，则返回true，否则返回false

s.issuperset(t)：如果t是s的超集，则返回true，否则返回false

s.union(t)：返回一个新的集合，该集合是s和t的并集

s.intersection(t)：返回一个新的集合，该集合是s和t的交集

s.difference(t)：返回一个新集合，该集合是s的成员，但不是t的成员

s1=set('abc')#{'b', 'a', 'c'}

s2=set('bcd')#{'b', 'd', 'c'}

len(s1)#3

s1 | s2#取并集{'b', 'c', 'a', 'd'}

s1 & s2#取交集{'b', 'c'}

s1 - s2#取差补，s1中有，s2中没有{'a'}

s1.add(10)

s1.update(['hello','world'])

s1.remove(10)

s1.issubset(s2)#s1是s2的子集吗？

s1.issuperset(s2)#s1是s2的超集吗？

s1.union(s2)#等于s1 | s2

s1.intersection(s2)#等于s1 &s2

s1.difference(s2)#等于s1 - s2

**时间方法**

time模块

时间表示方法

时间戳timestamp：表示的是从1970年1月1日00:00:00开始按秒计算的偏移量

UTC也是格林威治天文时间，世界标准时间。在中国为UTC+8。DST即夏令时

元组：由9个元素组成

struct\_time元组

tm\_year 2000

tm\_mon 1-12

tm\_mday 1-31

tm\_hour 0-23

tm\_min 0-59

tm\_sec 0-61

tm\_wday 0-6（0表示周一）

tm\_yday（一年中的第几天）1-366

tm\_isdst（是否为dst时间）默认为-1

time模块方法

time.localtime([secs])：将一个时间戳转换为当前时区的struct\_time。secs参数未提供，则以当前时间为准

time.gmtime([secs])：和localtime()方法类似，gmtime()方法是将一个是将一个时间戳转换为UTC时区（0时区）的struct\_time

time.time()：返回当前时间的时间戳

time.mktime(t)：将一个struct\_time转换为时间戳

time.sleep(secs)：线程推迟指定时间运行。单位为秒

time.astime([t])：把一个表示时间的元组或者struct\_time表示为这种形式：’sun jun 20 23:21:05 1993’。如果没有参数，将会将time.localtime()作为参数传入

time.ctime([secs])：把一个时间戳（按秒计算的浮点数）转换为time.asctime()的形式

time.strftime(format[,t])：把一个表示时间的元组或者struct（如由time.localtime()和time.gmtime()返回）转换为格式化的时间字符串。如果t未指定，将传入time.localtime()

time.strptime(string[,format])：把一个格式化时间字符串转换为struct\_time。实际上它和strftime()是逆操作

时间样式

%a本地简化星期名称

%A本地完整星期名称

%b本地简化月份名称

%B本地完整月份名称

%c本地相应的日期和时间

%d一个月中的第几天（01-31）

%H一天中的第几个小事（24小时制，00-23）

%I第几个小时（12小时制，01-12）

%j一年中的第几天（001-366）

%Z时区的名字

%m月份（01-12）

%M分钟数（00-59）

%p本地am或者pm的相应符

%s秒（01-61）

%U一年中的星期数（00-53，星期日是一个星期的开始）

%w一个星期中的第几天（0-6，0是星期天）

%x本地相应日期

%X本地相应时间

%y去掉世纪的年份

%Y完整的年份

import time

t1=time.localtime()#返回九元组，东八区时间time.struct\_time(tm\_year=2019, tm\_mon=10, tm\_mday=24, tm\_hour=20, tm\_min=57, tm\_sec=6, tm\_wday=3, tm\_yday=297, tm\_isdst=0)

time.gmtime()#0时区的九元组，格林威治时间time.struct\_time(tm\_year=2019, tm\_mon=10, tm\_mday=24, tm\_hour=12, tm\_min=58, tm\_sec=16, tm\_wday=3, tm\_yday=297, tm\_isdst=0)

time.time()#返回时间戳，常用1571922027.980559

time.mktime(t1)#把九元组转换成时间戳1571922122.0

time.sleep(1)#等待1秒

time.asctime()#默认返回当前的UTC+8时间'Thu Oct 24 21:03:31 2019'

time.ctime()#默认返回当前的UTC+8时间，常用'Thu Oct 24 21:03:31 2019'

time.ctime(0)#时间戳作为参数

time.asctime(t1)#元组作为参数

time.strftime('%Y-%m-%d %H:%M:%S')#常用'2019-10-24 21:06:50'

time.strptime('2018-08-20','%Y-%m-%d')#time.struct\_time(tm\_year=2018, tm\_mon=8, tm\_mday=20, tm\_hour=0, tm\_min=0, tm\_sec=0, tm\_wday=0, tm\_yday=232, tm\_isdst=-1)

datetime模块

datetime模块方法

datetime.today()：返回一个当前本地时间datetime对象

datetime.now([tz])：返回一个当前时间的datetime对象，如果提供了参数tz，则获取tz参数所指时区的本地时间

datetime.strptime(date\_string,format)：将格式字符串转换为datetime对象

datetime.ctime(datetime对象)：返回时间格式字符串

datetime.strftime(format)：返回指定格式字符串

时间计算

使用timedalta可以很方便的在日期上做days，小时hour，分钟，秒，毫秒，微秒的时间计算

from datetime import datetime,timedelta

t1 = datetime.now()#年月日时分秒毫秒datetime.datetime(2019, 10, 24, 21, 19, 16, 103642)，参数可以加时区参数

t1.year#2019

t1.month#10

t2=datetime.today()#datetime.datetime(2019, 10, 24, 21, 22, 8, 964971)

datetime.strptime('2019-10-24','%Y-%m-%d')#返回datetime对象，datetime.datetime(2019, 10, 24, 0, 0)

dt=timedelta(days=100)#设定100天

t1-dt#100天前datetime.datetime(2019, 7, 16, 21, 19, 16, 103642)

t1+dt#100天后datetime.datetime(2020, 2, 1, 21, 19, 16, 103642)

**异常处理**

python中的异常

当程序运行时，因为遇到未解的错误而导致中止运行，便会出现traceback小溪，打印异常

NameError未申明/初始化对象

IndexError序列中没有索引

SyntaxError语法错误

Keyboardinterrupt用户中断执行

EOFError没有内建输入，达到EOF标记

IOError输入/输出操作错误

try-except语句

定义了进行异常监控的一段代码，并且提供了处理异常的机制

try:

try\_suit#监控这里的异常

except filenotfounderror

except\_suite##异常处理代码

异常参数

异常也可以有参数，异常发生后会被传递给异常处理器

当异常被引发后参数是座位附加帮助信息传递给异常处理器

else子句

在try范围内没有被异常检测时，执行else子句

在else范围内的任何代码运行前，try范围内的所有代码必须完全成功

finally子句

finally子句是无论异常是否发生，是否捕捉都会执行一段代码

如果打开文件后，因为发生异常导致文件没有关闭，可能会发生数据损坏。使用finally可以保证文件总是能正常关闭。

try:

num=int(input('number:'))

result=100/num

print(num)

except(ValueError,ZeroDivisionError):

print('输入错误')

except(KeyboardInterrupt,EOFError):#写在一行用圆括号括起来，中间用逗号分隔

print('\nbyebye')

else:

print(result)#不发生异常才执行的代码

finally:

print('done')#不管是否发生都要执行的代码

#不是必须把所有语句写全，常用的又try-except和try-finally组合

触发异常

raise子句

遥想引发异常，最简单的形式就是输入关键字raise，后面跟药引发的异常名称，执行语句时，python会创建指定的异常类的一个对象，raise语句还可以指定异常对象进行初始化的参数

断言

断言时一句必须等价于布尔值为真的判定，此外，发生异常也意味着表达式为假

def set\_age(name,age):

if not 0 < age < 120:

raise ValueError('年龄超过范围')#rasie引发异常

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

def set\_age2(name,age):

assert 0 < age <120,'年龄超过范围'#断言异常，如果年龄在1到120之间，就输出年龄超出范围

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

**os相关模块**

os模块

模块简介

对文件系统的访问大多通过python的os模块实现，该模块主要是python访问操作系统功能的主要接口，有些方法，如copy等，并没有提供，可以使用shutil模块作为补充

import os

os.getcwd() # pwd

os.listdir() # ls

os.listdir('/tmp/mydemo') # ls /tmp/mydemo

os.mkdir('/tmp/mydemo')

os.chdir('/tmp/mydemo') # 切换目录

os.mknod('mytest') # touch mytest

os.symlink('/etc/hosts', 'zhuji') # ln -s /etc/hosts zhuji

os.makedirs('aaa/bbb/ccc') # mkdir -p aaa/bbb/ccc

os.path.isfile('zhuji') # 判断是文件吗

os.path.isdir('/abc') # 判断是目录吗

os.path.exists('/abc') # 判断是否存在，不管是文件还是目录

os.path.abspath('zhuji') # 显示绝对路径

os.path.basename('/tmp/abc/aaa.txt') # 显示最后一项

os.path.dirname('/tmp/aaa/aaa.txt') # 显示目录

os.path.split('/tmp/abc/aaa.txt') # 截成元组的形式

os.path.splitext('/tmp/abc/aaa.txt') # 截取扩展

os.path.join('tmp', 'abc') # 将两个拼接在一起

os.remove('mytest') # 删除文件

os.rmdir('aaa/bbb/ccc') # 删除空目录

pickle模块

pickle模块简介

将数据写入文件时，常规的文件方法只能把字符串对象写入。其他数据需先转成字符串再写入文件。python提供了一个标准的模块，称为pickle。使用它可以存在一个文件中存储人和python对象，之后可以完整无缺的取出来

pickle模块方法

分别调用dump()和load()可以存储、写入

import pickle as p

# pickle存储器可以将任意的数据类型写入文件，还可以无损的读出

shop\_file = '/tmp/shop.txt'

shop\_list = ['apple', 'eggs', 'banana']

with open(shop\_file, 'wb') as fobj:

p.dump(shop\_list, fobj) # 将列表写入文件

with open(shop\_file, 'rb') as fobj:

mylist = p.load(fobj) # 从文件中读取的数据仍然是列表

mylist[1]

**函数基础**

创建函数

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关键字组成，函数名字，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符串，和必须的函数体

前向引用

函数不允许在函数申明之前对齐进行引用或者调用

内部函数

在函数体内创建另外一个函数是完全合法的，这种函数叫做内部/内嵌函数

调用函数

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()

关键字参数

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 get\_info(name,age):

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

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

get\_info('bob',25)

get\_info(25,'bob')#能运行，语义不对

get\_info(age=25,name='bob')

# get\_info()#error

# get\_info('bob',25,30)#error

# get\_info(age=25,'bob')#key=val的形式必须在后面

# get\_info(25,name='bob')#25按参数顺序传递给name，又赋值bob给name，报错

get\_info('bob',age=25)

参数组

python允许程序员执行一个没有显示定义参数的函数

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

def func1(\*args):#表示args是个元组

print(args)

def func2(\*\*kwargs):#表示kwargs是个字典

print(kwargs)

def func3(\*args,\*\*kwargs):

print(args,kwargs)

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

func1()

func1('hao')

func1('hao',123)

func1('hao',123,'hello')

func2()

func2(name='zhangsan')

func2(name='zhangsan',age=23)

func3(10,'ehllo',name='bobo',age=20)

匿名函数

lambda

python允许用lambda关键字创造匿名函数

lambda x,y:x+y#表示有两个参数x,y，执行x+y

filter(func,seq)：调用一个布尔函数fun来迭代遍历每个序列的元素；返回一个使func返回值为true的序列

from random import randint

def func1(x):

return x %2

def func2(x):

return x \*2 +1

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

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

print(nums)

#filter的第一个参数是函数，这种函数叫做高阶函数

#将nums列表中的每一项交给func1处理，返回true保留，返回false过滤掉

print(list(filter(func1,nums)))

print(list(filter(lambda x:x %2,nums)))

print(list(map(func2,nums)))#将nums列表的每个数字交给func2加工

print(list(map(lambda x:x \*2+1,nums)))

**函数的高级应用**

变量作用域

全局变量：在一个模块中最高级别的变量有全局作用域

局部变量：局部变量只是暂时存在，仅仅只依赖于定义他们定义他们的函数现阶段是否处于活跃状态

global语句：局部变量想要覆盖全局变量使用global语句

x=20

print(x)#20

def bar():

global x

x=200

bar()

print(x)#200

名字函数

任何时候，总有一到三个活动的作用域（内建、全局和局部）

标识符的搜索顺序依次是局部、全局和内建

函数式编程

偏函数

一个带有多个参数的函数，如果其中某些参数基本上是固定的，那么就可以通过偏函数位这些参数赋值

from functools import partial

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

return a+b+c+d

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

print(add(10,20,30,5))

print(add(10,20,30,15))

print(add(10,20,30,25))

print(add(10,20,30,35))

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

print(myadd(5))

print(myadd(15))

print(myadd(25))

print(myadd(35))

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调用

def func(x):

if x==1:

return x

return x \* func(x-1)

#3\*func(2)

#3\*2\*func(1)

#3\*2\*1

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

print(func(4))

生成器

yield返回一个值给调用者并暂停执行，可以返回多个值

def mygen():

yield 'hello world!'

a=10+20

yield a

yield [10,20]

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

for i in mygen():

print(i)

#输出结果是三个值，是yield返回的值，并且生成器的返回值只执行一次，相当于返回多次的函数

内部函数

闭包（函数调用函数）

闭包将内部函数自己的代码和作用域以及外部函数的作用结合起来

import tkinter

from functools import partial

def hello(world):

def welcome():

lb.config(text='hello %s!' % world)

return welcome

root=tkinter.Tk()

lb=tkinter.Label(text='hello world',font="Times 26")

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

b1=mybth(text='Button1',command=hello('china'))

b2=mybth(text='Button2',command=hello('tedu'))

b3=mybth(text='quit',command=root.quit)

lb.pack()

b1.pack()

b2.pack()

b3.pack()

root.mainloop()

装饰器

装饰器是在函数调用之上的修饰，这些修饰仅是当申明一个函数或者方法的时候，才会应用的额外调用

def color(func):

def red():

return '\033[31;1m%s\033[0m' % func()

return red

def hello():

return 'Hello world!'

@color

def welcome():

return 'hello china!'

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

hello = color(hello) # 此种写法可以换成为hello加上@color

print(hello())

print(welcome()) # welcome因为有装饰器，所以调用时不是调用welcome函数，而是相当于color(welcome)()

# color(welcome)返回red,color(welcome)()

# 等于red()